Contents

Revision History ................................................................. 14

About this Document ...................................................... 15
   Intended Audience .................................................. 15
   Conventions ......................................................... 15
   Abbreviations ..................................................... 15
   Terminology ....................................................... 16

Chapter 1     Introduction ............................................... 17
   1.1 Overview .................................................... 17
   1.2 Specification ............................................... 18
      1.2.1 Processors ........................................... 18
      1.2.2 Operating Systems ................................. 18
      1.2.3 Compilers and Application Environment .......... 18
   1.3 Installing AMD uProf ...................................... 19
      1.3.1 Windows ............................................... 19
      1.3.2 Linux ................................................. 19
      1.3.3 FreeBSD .............................................. 21
   1.4 Sample Programs ............................................ 21
   1.5 Support ..................................................... 22

Chapter 2     Workflow and Key Concepts ............................. 23
   2.1 Workflow .................................................... 23
      2.1.1 Collect Phase ....................................... 23
      2.1.2 Translate Phase ...................................... 25
      2.1.3 Analyze Phase ...................................... 25
   2.2 Predefined Sampling Configuration ....................... 25
   2.3 Predefined View Configuration ............................ 26

Chapter 3     Getting started with AMDuProfPcm – System Analysis .......................... 30
   3.1 Options ..................................................... 30
   3.2 Examples ................................................... 38
      3.2.1 Linux and FreeBSD ................................. 38
Contents

6.3 Collect Command ................................................................. 71
  6.3.1 Options ................................................................. 71
  6.3.2 Windows Specific Options ........................................... 73
  6.3.3 Linux Specific Options .............................................. 74
  6.3.4 Examples ............................................................... 75
6.4 Report Command ............................................................... 78
  6.4.1 Options ................................................................. 79
  6.4.2 Windows Specific Options ........................................... 81
  6.4.3 Linux Specific Options .............................................. 81
  6.4.4 Examples ............................................................... 82
6.5 Timechart Command .......................................................... 83
  6.5.1 Options ................................................................. 84
  6.5.2 Examples ............................................................... 84
6.6 Info Command ................................................................. 85
  6.6.1 Options ................................................................. 86
  6.6.2 Examples ............................................................... 87

Chapter 7 Performance Analysis ............................................... 88
  7.1 CPU Profiling ............................................................... 88
  7.2 Analysis with Time-based Profiling ................................... 89
    7.2.1 Configuring and Starting Profile ................................ 89
    7.2.2 Analyzing Profile Data .......................................... 90
  7.3 Analysis with Event-based Profiling ................................ 91
    7.3.1 Configuring and Starting Profile ................................ 91
    7.3.2 Analyzing Profile Data .......................................... 92
  7.4 Analysis with Instruction-based Sampling ........................... 92
    7.4.1 Configuring and Starting Profile ................................ 92
    7.4.2 Analyzing Profile Data .......................................... 93
  7.5 Analysis with Call Stack Samples .................................... 93
    7.5.1 Flame Graph ...................................................... 94
    7.5.2 Call Graph ...................................................... 95
  7.6 Profiling a Java Application ............................................ 96
7.6.1 Launching a Java Application ................................................. 96
7.6.2 Attaching a Java Process to Profile ........................................ 97
7.6.3 Java Source View ............................................................... 97
7.6.4 Java Call Stack Profile ........................................................ 98
7.7 Cache Analysis ................................................................. 99
  7.7.1 Supported Metrics .......................................................... 100
  7.7.2 Cache Analysis Using GUI ............................................... 100
  7.7.3 Cache Analysis Using CLI ............................................... 101
7.8 Custom Profile ................................................................. 104
  7.8.1 Configuring and Starting Profile ....................................... 104
  7.8.2 Analyzing Profile Data .................................................. 105
7.9 Advisory ........................................................................ 105
  7.9.1 Confidence Threshold ................................................... 105
  7.9.2 Issue Threshold ............................................................ 106
7.10 ASCII Dump of IBS Samples ............................................... 106
7.11 Limitations ................................................................. 107

Chapter 8 Performance Analysis (Linux) ....................................... 108

  8.1 OpenMP Analysis ............................................................. 108
    8.1.1 Profiling OpenMP Application using GUI .......................... 109
    8.1.2 Profiling OpenMP Application Using CLI ......................... 110
    8.1.3 Environment Variables ............................................... 111
    8.1.4 Limitations ............................................................. 111
  8.2 MPI Profiling ................................................................. 112
    8.2.1 Collecting Data Using CLI ........................................... 112
    8.2.2 Analyzing the Data with CLI ...................................... 114
    8.2.3 Analyze the Data with GUI ......................................... 114
    8.2.4 Limitations ............................................................. 114
  8.3 Profiling Linux System Modules .......................................... 115
  8.4 Profiling Linux Kernel ..................................................... 115
    8.4.1 Enabling Kernel Symbol Resolution ............................... 115
    8.4.2 Downloading and Installing Kernel Debug Symbol Packages .... 116
8.4.3 Build Linux kernel with Debug Symbols ...........................................117
8.4.4 Analyzing Hotspots in Kernel Functions ...........................................117
8.4.5 Linux Kernel Callstack Sampling .....................................................117
8.4.6 Constraints ..................................................................................118

8.5 Holistic Analysis ..............................................................................118
8.5.1 Holistic Analysis Using GUI ..............................................................120
8.5.2 Holistic Analysis Using CLI ..............................................................123

8.6 Thread State Analysis .......................................................................125
8.6.1 Thread State Analysis Using CLI .......................................................127

8.7 Kernel Block I/O Analysis .................................................................129
8.7.1 Kernel Block I/O Analysis Using CLI ...............................................129
8.7.2 Kernel Block I/O Analysis Using GUI ...............................................131

8.8 GPU Offloading Analysis (GPU Tracing) ...........................................132
8.8.1 GPU Offload Analysis Using GUI ....................................................132
8.8.2 GPU Offload Analysis Using CLI ....................................................135

8.9 GPU Profiling .................................................................................137
8.9.1 GPU Profiling Using CLI .................................................................139

8.10 Other OS Tracing Events ..............................................................140
8.10.1 Tracing Page Faults and Memory Allocations Using CLI .................141

8.11 MPI Trace Analysis .......................................................................142
8.11.1 MPI Light-weight Tracing Using CLI ..............................................143
8.11.2 MPI Full Tracing Using CLI ...........................................................144
8.11.3 MPI FULL Tracing Using GUI .........................................................148

Chapter 9 Power Profile .......................................................................150
9.1 Metrics ............................................................................................150
9.2 Using Profile through GUI ...............................................................152
   9.2.1 Configuring a Profile ................................................................153
   9.2.2 Analyzing a Profile ..................................................................154
9.3 Using CLI to Profile ........................................................................155
   9.3.1 Examples ..................................................................................156
9.4 AMDPowerProfileAPI Library ..........................................................156

Contents
Contents

9.4.1 Using the APIs ..................................................157
9.5 Limitations .........................................................157

Chapter 10 Energy Analysis .............................................158
10.1 Using CLI to Profile .............................................158
10.2 Limitations .........................................................159

Chapter 11 Profile Control APIs .......................................160
11.1 AMDProfileControl APIs ........................................160
  11.1.1 Profile Control APIs ........................................160
  11.1.2 Using the APIs ..............................................161
  11.1.3 Compiling Instrumented Target Application ..................162
  11.1.4 Profiling Instrumented Target Application ....................162

Chapter 12 Reference ....................................................163
12.1 Preparing an Application for Profiling .........................163
  12.1.1 Generating Debug Information on Windows ..................163
  12.1.2 Generating Debug Information on Linux ....................164
12.2 CPU Profiling .....................................................164
  12.2.1 Hardware Sources ...........................................165
  12.2.2 Profiling Concepts ..........................................166
  12.2.3 Profile Types ................................................167
  12.2.4 Predefined Core PMC Events ...............................168
  12.2.5 IBS Derived Events .........................................176
12.3 Useful URLs ......................................................182
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Conventions</td>
<td>15</td>
</tr>
<tr>
<td>Table 2</td>
<td>Abbreviations</td>
<td>15</td>
</tr>
<tr>
<td>Table 3</td>
<td>Terminology</td>
<td>16</td>
</tr>
<tr>
<td>Table 4</td>
<td>User Interface</td>
<td>17</td>
</tr>
<tr>
<td>Table 5</td>
<td>Sampled Data</td>
<td>24</td>
</tr>
<tr>
<td>Table 6</td>
<td>Predefined Sampling Configurations</td>
<td>25</td>
</tr>
<tr>
<td>Table 7</td>
<td>Assess Performance Configurations</td>
<td>26</td>
</tr>
<tr>
<td>Table 8</td>
<td>Investigate Data Access Configurations</td>
<td>27</td>
</tr>
<tr>
<td>Table 9</td>
<td>Investigate Branch Access Configurations</td>
<td>27</td>
</tr>
<tr>
<td>Table 10</td>
<td>Assess Performance (Extended) Configurations</td>
<td>28</td>
</tr>
<tr>
<td>Table 11</td>
<td>Investigate Instruction Access</td>
<td>28</td>
</tr>
<tr>
<td>Table 12</td>
<td>Instruction Based Sampling Configurations</td>
<td>28</td>
</tr>
<tr>
<td>Table 13</td>
<td>AMDuProfPcm Options</td>
<td>30</td>
</tr>
<tr>
<td>Table 14</td>
<td>Performance Metrics 1</td>
<td>33</td>
</tr>
<tr>
<td>Table 15</td>
<td>Performance Metrics 2</td>
<td>35</td>
</tr>
<tr>
<td>Table 16</td>
<td>AMDuProfSys Collect Command Options</td>
<td>42</td>
</tr>
<tr>
<td>Table 17</td>
<td>AMDuProfSys Report Command Options</td>
<td>43</td>
</tr>
<tr>
<td>Table 18</td>
<td>Supported Commands</td>
<td>67</td>
</tr>
<tr>
<td>Table 19</td>
<td>AMDuProfCLI Collect Command Options</td>
<td>71</td>
</tr>
<tr>
<td>Table 20</td>
<td>Collect Command – Windows Specific Options</td>
<td>73</td>
</tr>
<tr>
<td>Table 21</td>
<td>Collect Command – Linux Specific Options</td>
<td>74</td>
</tr>
<tr>
<td>Table 22</td>
<td>AMDuProfCLI Report Command Options</td>
<td>79</td>
</tr>
<tr>
<td>Table 23</td>
<td>AMDuProfCLI Report Command Windows Specific Options</td>
<td>81</td>
</tr>
<tr>
<td>Table 24</td>
<td>AMDuProfCLI Report Command Linux Specific Options</td>
<td>81</td>
</tr>
<tr>
<td>Table 25</td>
<td>Timechart Command Options</td>
<td>84</td>
</tr>
<tr>
<td>Table 26</td>
<td>Info Command Options</td>
<td>86</td>
</tr>
<tr>
<td>Table 27</td>
<td>Info Command - Linux Specific Options</td>
<td>86</td>
</tr>
<tr>
<td>Table 28</td>
<td>IBS OP Derived Metrics</td>
<td>100</td>
</tr>
<tr>
<td>Table 29</td>
<td>Sort-by Metric</td>
<td>103</td>
</tr>
</tbody>
</table>
Table 30. Support Matrix ................................................................. 108
Table 31. MPI Profiling Support Matrix ............................................. 112
Table 32. Supported Events for Holistic Analysis .............................. 119
Table 33. I/O Operations ............................................................... 129
Table 34. Supported Interfaces for GPU Tracing .............................. 132
Table 35. Supported Events for GPU Profiling ................................. 137
Table 36. Supported Metrics for GPU Profiling ............................... 138
Table 37. Supported Events for OS Tracing ...................................... 140
Table 38. List of Supported MPI APIs for Light-weight Tracing ........... 143
Table 39. MPI APIs ................................................................. 145
Table 40. Family 17h Model 00h – 0Fh (AMD Ryzen™, AMD Ryzen ThreadRipper™, and 1st Gen AMD EPYC™) 150
Table 41. Family 17h Model 10h — (AMD Ryzen™ and AMD Ryzen™ PRO APU) 151
Table 42. Family 17h Model 70h — 7Fh (3rd Gen AMD Ryzen™) .......... 151
Table 43. Family 17h Model 30h — 3Fh (EPIC 7002) ........................ 151
Table 44. Supported Counter Categories for Older APU Families ......... 152
Table 45. Supported Categories for dGPUs ................................. 152
Table 46. Predefined Core PMC Events ............................................ 168
Table 47. Core CPU Metrics .......................................................... 176
Table 48. IBS Fetch Events .............................................................. 177
Table 49. IBS Op Events ............................................................... 178
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.</td>
<td>AMD uProf GUI</td>
<td>45</td>
</tr>
<tr>
<td>Figure 2.</td>
<td>AMD uProf Welcome Screen</td>
<td>46</td>
</tr>
<tr>
<td>Figure 3.</td>
<td>Start Profiling - Select Profile Target</td>
<td>48</td>
</tr>
<tr>
<td>Figure 4.</td>
<td>Start Profiling - Select Profile Type</td>
<td>49</td>
</tr>
<tr>
<td>Figure 5.</td>
<td>Start Profiling - Advanced Options 1</td>
<td>50</td>
</tr>
<tr>
<td>Figure 6.</td>
<td>Start Profiling - Advanced Options 2</td>
<td>51</td>
</tr>
<tr>
<td>Figure 7.</td>
<td>Profile Data Collection</td>
<td>52</td>
</tr>
<tr>
<td>Figure 8.</td>
<td>Summary - Hot Spots Screen</td>
<td>53</td>
</tr>
<tr>
<td>Figure 9.</td>
<td>Summary - Thread Concurrency Graph</td>
<td>54</td>
</tr>
<tr>
<td>Figure 10.</td>
<td>ANALYZE - Function HotSpots</td>
<td>55</td>
</tr>
<tr>
<td>Figure 11.</td>
<td>Analyze - Metrics</td>
<td>56</td>
</tr>
<tr>
<td>Figure 12.</td>
<td>SOURCES - Source and Assembly</td>
<td>58</td>
</tr>
<tr>
<td>Figure 13.</td>
<td>ANALYZE - Flame Graph</td>
<td>59</td>
</tr>
<tr>
<td>Figure 14.</td>
<td>ANALYZE - Call Graph</td>
<td>61</td>
</tr>
<tr>
<td>Figure 15.</td>
<td>Import Session – Importing Profile Database</td>
<td>62</td>
</tr>
<tr>
<td>Figure 16.</td>
<td>PROFILE - Recent Session(s)</td>
<td>63</td>
</tr>
<tr>
<td>Figure 17.</td>
<td>PROFILE - Saved Configurations</td>
<td>64</td>
</tr>
<tr>
<td>Figure 18.</td>
<td>SETTINGS - Preferences</td>
<td>65</td>
</tr>
<tr>
<td>Figure 19.</td>
<td>SETTINGS - Symbols</td>
<td>66</td>
</tr>
<tr>
<td>Figure 20.</td>
<td>SETTINGS - Source Data</td>
<td>66</td>
</tr>
<tr>
<td>Figure 21.</td>
<td>Collect and Report Commands</td>
<td>68</td>
</tr>
<tr>
<td>Figure 22.</td>
<td>List Supported Predefined Configurations</td>
<td>69</td>
</tr>
<tr>
<td>Figure 23.</td>
<td>Output of timechart --list Command</td>
<td>70</td>
</tr>
<tr>
<td>Figure 24.</td>
<td>Execution of timechart</td>
<td>71</td>
</tr>
<tr>
<td>Figure 25.</td>
<td>Time-based Profile – Configure</td>
<td>90</td>
</tr>
<tr>
<td>Figure 26.</td>
<td>Event-based Profile – Configure</td>
<td>91</td>
</tr>
<tr>
<td>Figure 27.</td>
<td>Start Profiling - Advanced Options</td>
<td>94</td>
</tr>
<tr>
<td>Figure 28.</td>
<td>ANALYZE - Flame Graph</td>
<td>95</td>
</tr>
<tr>
<td>Figure 29.</td>
<td>ANALYZE - Call Graph</td>
<td>96</td>
</tr>
</tbody>
</table>
Figure 30. Java Method - Source View ................................................................. 98
Figure 31. Java Application - Flame Graph ......................................................... 99
Figure 32. Cache Analysis (from AMD uProf 3.4) ............................................. 101
Figure 33. Cache Analysis - Summary Sections (from AMD uProf 3.4) ............. 102
Figure 34. Cache Analysis - Detailed Report (from AMD uProf 3.4) ............... 103
Figure 35. Start Profiling - Custom Profile ......................................................... 104
Figure 36. CPI Metric - Threshold-based Performance Issue ............................ 106
Figure 37. Enable OpenMP Tracing ................................................................. 109
Figure 38. HPC - Overview (from AMD uProf 3.4) .......................................... 109
Figure 39. HPC - Parallel Regions (from AMD uProf 3.4) ............................... 110
Figure 40. An OpenMP Report ................................................................. 111
Figure 41. Advanced Options 1 ....................................................................... 120
Figure 42. Advanced Options 2 ....................................................................... 121
Figure 43. System Analysis - TIMECHART ...................................................... 122
Figure 44. System Analysis - OS Tracing Report ............................................. 124
Figure 45. Thread State Analysis - TIMECHART ........................................... 126
Figure 46. Thread State Analysis - OS Tracing Report .................................. 128
Figure 47. Disk I/O Summary Tables ............................................................... 130
Figure 48. ANALYZE - Block I/O Stats .......................................................... 131
Figure 49. GPU Tracing Options ................................................................. 133
Figure 50. TIMECHART - GPU Offload Analysis ........................................ 134
Figure 51. GPU Tracing Report ................................................................. 136
Figure 52. GPU Profile Report ....................................................................... 140
Figure 53. Monitored Events ................................................................. 142
Figure 54. LWT Report ............................................................................... 144
Figure 55. MPI Communicator Summary Table ............................................ 146
Figure 56. MPI Rank Summary Table ............................................................. 146
Figure 57. MPI Point-to-Point API Summary Table ...................................... 147
Figure 58. MPI Communication Matrix ............................................................ 147
Figure 59. MPI Collective API Summary Table ............................................ 147
Figure 60. Import Profile Session .................................................................. 148
Figure 61.  MPI Communication Matrix ................................................................. 148
Figure 62.  TIMECHART - MPI Rank Analysis ..................................................... 149
Figure 63.  Live System-wide Power Profile (from AMD uProf 3.4) .................... 153
Figure 64.  Timechart Page ................................................................. 154
Figure 65.  --list Command Output ................................................................. 155
Figure 66.  Timechart Run ................................................................. 155
Figure 67.  AMDTClassicMatMul Property Page ............................................. 164
# Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 2022</td>
<td>3.5</td>
<td>Included AMD uProf 3.5 features</td>
</tr>
<tr>
<td>April 2021</td>
<td>Initial</td>
<td>Documented AMD uProf 3.4 features</td>
</tr>
</tbody>
</table>
About this Document

This document describes how to use AMD uProf to perform CPU, GPU, and power analysis of applications running on Windows®, Linux®, and FreeBSD® operating systems on AMD processors. The latest version of this document is available in the AMD uProf web site (https://developer.amd.com/amd-uprof/).

Intended Audience

This document is intended for the software developers and performance tuning experts who want to improve the performance of their application. It assumes prior understanding of CPU architecture, concepts of threads, processes, load modules, and familiarity with performance analysis concepts.

Conventions

The following conventions have been used in this document:

<table>
<thead>
<tr>
<th>Table 1. Conventions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convention</td>
</tr>
<tr>
<td>GUI element</td>
</tr>
<tr>
<td>&gt;</td>
</tr>
<tr>
<td>[]</td>
</tr>
<tr>
<td>…</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>File name</td>
</tr>
<tr>
<td>Command</td>
</tr>
<tr>
<td>Hyperlink</td>
</tr>
</tbody>
</table>

Abbreviations

The following abbreviations have been used in this document:

<table>
<thead>
<tr>
<th>Table 2. Abbreviations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abbreviation</td>
</tr>
<tr>
<td>ASLR</td>
</tr>
<tr>
<td>CLI</td>
</tr>
<tr>
<td>CPI</td>
</tr>
<tr>
<td>CSV</td>
</tr>
</tbody>
</table>
Table 2. Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DC</td>
<td>Data Cache</td>
</tr>
<tr>
<td>DTLB</td>
<td>Data Translation Lookaside Buffer</td>
</tr>
<tr>
<td>EBP</td>
<td>Event Based Profiling, uses Core PMC events</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IBS</td>
<td>Instruction Based Sampling</td>
</tr>
<tr>
<td>IC</td>
<td>Instruction Cache</td>
</tr>
<tr>
<td>IPC</td>
<td>Instructions Per Cycle</td>
</tr>
<tr>
<td>ITLB</td>
<td>Instruction Translation Lookaside Buffer</td>
</tr>
<tr>
<td>MSR</td>
<td>Model Specific Register</td>
</tr>
<tr>
<td>NB</td>
<td>Northbridge</td>
</tr>
<tr>
<td>PMC</td>
<td>Performance Monitoring Counter</td>
</tr>
<tr>
<td>PTI</td>
<td>Per Thousand Instructions</td>
</tr>
<tr>
<td>RAPL</td>
<td>Running Average Power Limit</td>
</tr>
<tr>
<td>SMU</td>
<td>System Management Unit</td>
</tr>
<tr>
<td>TBP</td>
<td>Timer Based Profiling</td>
</tr>
</tbody>
</table>

Terminology

The following terms have been used in this document:

Table 3. Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD uProf</td>
<td>The product name uProf.</td>
</tr>
<tr>
<td>AMDuProf</td>
<td>The name of the graphical user interface tool.</td>
</tr>
<tr>
<td>AMDuProfCLI</td>
<td>The name of the command line interface tool.</td>
</tr>
<tr>
<td>AMDuProfPcm</td>
<td>The name of the command line interface tool for System Analysis.</td>
</tr>
<tr>
<td>AMDuProfSys</td>
<td>The name of the python based command line interface tool for System Analysis.</td>
</tr>
<tr>
<td>Client</td>
<td>Instance of AMD uProf or AMDuProfCLI running on a host system.</td>
</tr>
<tr>
<td>Host system</td>
<td>System in which the AMD uProf client process runs.</td>
</tr>
<tr>
<td>Performance Profiling (or) CPU Profiling</td>
<td>Identify and analyze the performance bottlenecks. Performance Profiling and CPU Profiling denotes the same.</td>
</tr>
<tr>
<td>System Analysis</td>
<td>Refers to AMDuProfPcm or AMDuProfSys tools.</td>
</tr>
<tr>
<td>Target system</td>
<td>System in which the profile data is collected.</td>
</tr>
</tbody>
</table>
Chapter 1  Introduction

1.1  Overview

AMD uProf is a performance analysis tool for applications running on Windows and Linux operating systems. It allows developers to understand and improve the runtime performance of their application.

AMD uProf offers the following functionalities:

• **Performance Analysis (CPU Profile)**
  To identify runtime performance bottlenecks of the application.

• **System Analysis**
  To monitor system performance metrics, such as IPC and memory bandwidth.

• **Live Power Profile**
  To monitor thermal and power characteristics of the system.

• **Energy Analysis**
  To identify energy hotspots in the application (Windows only).

AMD uProf has the following user interfaces:

**Table 4. User Interface**

<table>
<thead>
<tr>
<th>Executable</th>
<th>Description</th>
<th>Supported OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMDuProf</td>
<td>GUI to perform CPU and Power Profile</td>
<td>Windows and Linux</td>
</tr>
<tr>
<td>AMDuProfCLI</td>
<td>CLI to perform CPU and Power Profile</td>
<td>Windows, Linux, and FreeBSD</td>
</tr>
<tr>
<td>AMDuProfPcm</td>
<td>CLI to perform System Analysis</td>
<td>Windows, Linux, and FreeBSD</td>
</tr>
<tr>
<td>AMDPerf/AMDuProfSys.py</td>
<td>Python script for System Analysis</td>
<td>Windows and Linux</td>
</tr>
</tbody>
</table>

AMD uProf can effectively be used to:

• Analyze the performance of one or more processes/applications.
• Track down the performance bottlenecks in the source code.
• Identify ways to optimize the source code for better performance and power efficiency.
• Examine the behavior of kernels, drivers, and system modules.
• Observe system level thermal and power characteristics.
• Observe system metrics, such as IPC and memory bandwidth.
1.2 Specification

AMD uProf supports the following specifications. For a detailed list of supported processors and operating systems, refer AMD uProf Release Notes.

1.2.1 Processors

- AMD “Zen”-based CPU and APU Processors
- AMD Instinct™ MI100 and MI200 accelerators (for GPU kernel profiling and tracing)
- Intel® Processors (Timer based profiling only)

1.2.2 Operating Systems

AMD uProf supports the 64-bit versions of the following operating systems:

- Microsoft
  - Windows 10 and 11
  - Windows Server 2019 and 2022
- Linux
  - Ubuntu 16.04 and later
  - RHEL 7.0 and later
  - CentOS 7.0 and later
  - openSUSE Leap 15.0
  - SLES 12 and 15
- FreeBSD 12.2 and later

1.2.3 Compilers and Application Environment

AMD uProf supports the following application environments:

- Languages
  - Native languages: C, C++, Fortran, and Assembly
  - Non-native languages: Java and C#
- Programs compiled with
  - Microsoft compilers, GNU compilers, and LLVM
  - AMD Optimizing CPU Compilers (AOCC)
  - Intel Compilers (ICC)
• Parallelism
  – OpenMP
  – MPI
• Debug info formats: PDB, COFF, DWARF, and STABS
• Applications compiled with and without optimization or debug information
• Single-process, multi-process, single-thread, and multi-threaded applications
• Dynamically linked/loaded libraries
• POSIX development environment on Windows
  – Cygwin
  – MinGW

1.3 Installing AMD uProf

Download the latest version of the AMD uProf installer package for the supported operating systems from AMD Developer Central (https://developer.amd.com/amd-uprof/). You can install it using one of the following methods.

1.3.1 Windows

Run the 64-bit Windows installer binary AMDuProf-x.y.z.exe.

After the installation is complete, the executables, libraries, and the other required files are installed in the folder C:\Program Files\AMD\AMDuProf/.

1.3.2 Linux

Installing Using a tar File

Extract the tar.bz2 binary file and install AMD uProf using the following command:

$ tar -xf AMDuProf_Linux_x64_x.y.z.tar.bz2

*Note: The Power Profiler Linux Driver must be installed manually.*

Installing Using a RPM Package (RHEL)

Install the AMD uProf RPM package by using the rpm or yum command:

$ sudo rpm --install amduprof-x.y-z.x86_64.rpm
$ sudo yum install amduprof-x.y-z.x86_64.rpm

After the installation is complete, the executables, libraries, and the other required files will be installed in the directory /opt/AMDuProf_X.Y-ZZZ/.

Installing Using a Debian Package (Ubuntu)
Install the AMD uProf Debian package by using the `dpkg` command:

```bash
$ sudo dpkg --install amduprof_x.y-z_amd64.deb
```

After the installation is complete, the executables, libraries, and the other required files will be installed in the directory `/opt/AMDuProf_X.Y-ZZZ/`.

**Installing Power Profiling Driver on Linux**

While installing AMD uProf using RPM and Debian installer packages, the Power Profiler Linux Driver build is generated and installed automatically. However, if you downloaded the AMD uProf tar.bz2 archive, you must install the Power Profiler Linux Driver manually.

The GCC and MAKE software packages are prerequisites for installing Power Profiler Driver. If you do not have these packages, you can install them using the following commands:

On RHEL and CentOS distros:

```bash
$ sudo yum install gcc make
```

On Debian/Ubuntu distros:

```bash
$ sudo apt install build-essential
```

Execute the following commands:

```bash
$ tar -xf AMDuProf_Linux_x64_x.y.z.tar.bz2
$ cd AMDuProf_Linux_x64_x.y.z/bin
$ sudo ./AMDPowerProfilerDriver.sh install
```

Installer will create a source tree for Power Profiler Driver in the directory `/usr/src/AMDPowerProfiler-<version>`. All the source files required for module compilation are in this directory and under MIT license.

To uninstall the driver run the following commands:

```bash
$ cd AMDuProf_Linux_x64_x.y.z/bin
$ sudo ./AMDPowerProfilerDriver.sh uninstall
```

**Linux Power Profiling Driver Support for DKMS**

On Linux machines, Power profiling driver can also be installed with Dynamic Kernel Module Support (DKMS) framework support. DKMS framework automatically upgrades the Power Profiler Driver module whenever there is a change in the existing kernel. This saves you from manually upgrading the power profiling driver module. The DKMS package must be installed on target machines before running the installation steps mentioned in the above section. `AMDPowerProfilerDriver.sh` installer script will automatically handle the DKMS related configuration if the DKMS package is installed on the target machine.

Example (for Ubuntu distros):

```bash
$ sudo apt-get install dkms
$ tar -xf AMDuProf_Linux_x64_x.y.z.tar.bz2
$ cd AMDuProf_Linux_x64_x.y.z/bin
$ sudo ./AMDPowerProfilerDriver.sh install
```

If you upgrade the kernel version frequently, it is recommended to use DKMS for the installation.
Installing ROCm

Complete the steps in the ROCm installation guide (https://rocmdocs.amd.com/en/latest/Installation_Guide/Installation-Guide.html) to install AMD ROCm™ v4.3 on the host system.

After ROCm 4.3.0 installation, make sure symbolic link of /opt/rocm/ points to /opt/rocm-4.3.0/.

$ ln -s /opt/rocm-4.3.0/ /opt/rocm/

AMD ROCm v4.3 installation is required for GPU tracing and profiling.

Installing BCC and eBPF

Complete the steps on the BCC website (https://github.com/iovisor/bcc/blob/master/INSTALL.md) to install it.

After installing BCC, run the following command to validate the BCC installation:

$ cd AMDuProf_Linux_x64_x.y.z/bin
$ sudo ./AMDuProfVerifyBpfInstallation.sh

If you install AMD uProf using RPM/DEB installer, the script is run by the installer and the info about BCC installation and eBPF (Extended Berkeley Packet Filter) support on the host is provided.

1.3.3 FreeBSD

Extracting the tar.bz2 binary file and install AMD uProf:

$ tar -xf AMDuProf_FreeBSD_x64_x.y.z.tar.bz2

1.4 Sample Programs

A few sample programs are installed along with the product for you to use with the tool:

- Windows
  A sample matrix multiplication application
  C:\Program Files\AMD\AMDuProf\Examples\AMDTClassicMatMul\bin\AMDTClassicMatMul.exe

- Linux
  - A sample matrix multiplication program with makefile
    /opt/AMDuProf_X.Y-ZZZ/Examples/AMDTClassicMat/
  - An OpenMP example program and its variants with makefile
    /opt/AMDuProf_X.Y-ZZZ/Examples/CollatzSequence_C-OMP/
• FreeBSD

A sample matrix multiplication program with makefile

/\install\dir>/AMDuProf_FreeBSD_x64_X.Y.ZZZ/Examples/AMDTClassicMat/

1.5 Support

For support options, the latest documentation, and downloads refer AMD Developer Central (https://developer.amd.com/amd-uprof/).
Chapter 2     Workflow and Key Concepts

2.1     Workflow

The AMD uProf workflow has the following phases:
1. Collect — Run the application program and collect the profile data.
2. Translate — Process the profile data to aggregate, correlate, and organize into database.
3. Analyze — View and analyze the performance data to identify the bottlenecks.

2.1.1     Collect Phase

Important concepts of the collect phase are explained in this section.

Profile Target

The profile target is one of the following for which profile data will be collected:
• Application — Launch application and profile that process and its children.
• System — Profile all the running processes and/or kernel.
• Process — Attach to a running application (native applications only).

Profile Type

The profile type defines the type of profile data collected and how the data should be collected. The following profile types are supported:
• CPU Profile
• GPU Profile
• GPU Trace
• System-wide Power Profile
• Power Application Analysis (Windows only)

The data collection is defined by Sampling Configuration:
• Sampling Configuration identifies the set of Sampling Events, their Sampling Interval, and mode.
• Sampling Event is a resource used to trigger a sampling point at which a sample (profile data) will be collected.
• Sampling Interval defines the number of the occurrences of the sampling event after which an interrupt will be generated to collect the sample.
• **Mode** defines when to count the occurrences of the sampling event – in User mode and/or OS mode.

Type of profile data to collect – **Sampled Data**:

**Sampled Data** — the profile data that can be collected when the interrupt is generated (upon the expiry of the sampling interval of a sampling event).

The following table shows the type of profile data collected and sampling events for a profile type:

<table>
<thead>
<tr>
<th>Table 5. Sampled Data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Profile Type</strong></td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>CPU Profiling</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>GPU Profiling</td>
</tr>
<tr>
<td>GPU Tracing</td>
</tr>
</tbody>
</table>

For CPU Profiling, there are numerous micro-architecture specific events available to monitor. The tool groups the related and interesting events to monitor called **Predefined Sampling Configuration**. For example, **Assess Performance** is one such configuration used to get the overall assessment of the performance and to find potential issues for investigation. For more information, refer “Predefined View Configuration” on page 26.

A **Custom Sampling Configuration** is the one in which you can define a sampling configuration with events of interest.

**Profile Configuration**

A profile configuration identifies all the information used to collect the measurement. It contains the information about profile target, sampling configuration, data to sample, and profile scheduling details.

The GUI saves these profile configuration details with a default name (for example, AMDuProf-TBP-Classic), you can define them too. As the performance analysis is iterative, this is persistent (can be deleted) and hence, you can also reuse the same configuration for the future data collection runs.

**Profile Session (or Profile Run)**

A profile session represents a single performance experiment for a profile configuration. The tool saves all the profile and translated data (in a database) in the folder named as `<profile config name>-<timestamp>`.
Once the profile data is collected, the GUI processes the data to aggregate and attribute the samples to the respective processes, threads, load modules, functions, and instructions. This aggregated data is then written into an SQLite database used during the Analyze phase. This process of the translating the raw profile data happens in the CLI while generating the profile report.

2.1.2 Translate Phase

The collected raw profile data is processed to aggregate and attribute to the respective processes, threads, load modules, functions, and instructions. The debug information for the launched application generated by the compiler is needed to correlate the samples to functions and source lines.

This phase is performed automatically in the GUI after the profiling is stopped. In the CLI, you can invoke the report command to generate the report from the raw profile file.

2.1.3 Analyze Phase

View Configuration

A View is a set of sampled event data and computed performance metrics either displayed in the GUI pages or in the text report generated by the CLI. Each predefined sampling configuration has a list of associated predefined views.

The tool can be used to filter/view only specific configurations, which is called Predefined View. For example, IPC assessment view lists metrics such as CPU Clocks, Retired Instructions, IPC, and CPI. For more information, refer “Predefined Sampling Configuration” on page 25.

2.2 Predefined Sampling Configuration

The Predefined Sampling Configuration provides a convenient way to select a useful set of sampling events for profile analysis. The following table lists all such configurations:

<table>
<thead>
<tr>
<th>Profile Type</th>
<th>Predefined Configuration Name</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-based profile (TBP)</td>
<td>Time-based profile</td>
<td>tbp</td>
<td>To identify where the programs are consuming time.</td>
</tr>
</tbody>
</table>
Table 6. Predefined Sampling Configurations

<table>
<thead>
<tr>
<th>Profile Type</th>
<th>Predefined Configuration Name</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Assess performance (Extended)</td>
<td>assess_ext</td>
<td>Provides an overall assessment of the performance with additional metrics.</td>
</tr>
<tr>
<td></td>
<td>Investigate data access</td>
<td>data_access</td>
<td>To find data access operations with poor L1 data cache locality and poor DTLB behavior.</td>
</tr>
<tr>
<td></td>
<td>Investigate instruction access</td>
<td>inst_access</td>
<td>To find instruction fetches with poor L1 instruction cache locality and poor ITLB behavior.</td>
</tr>
<tr>
<td></td>
<td>Investigate branching</td>
<td>branch</td>
<td>To find poorly predicted branches and near returns.</td>
</tr>
<tr>
<td>IBS</td>
<td>Instruction based sampling</td>
<td>ibs</td>
<td>To collect the sample data using IBS Fetch and IBS OP. Precise sample attribution to instructions.</td>
</tr>
<tr>
<td></td>
<td>Cache Analysis</td>
<td>memory</td>
<td>To identify the false cache-line sharing issues. The profile data will be collected using IBS OP.</td>
</tr>
<tr>
<td>Energy</td>
<td>Power application analysis</td>
<td>power</td>
<td>To identify where the programs are consuming energy.</td>
</tr>
</tbody>
</table>

Notes:
1. The AMDuProf GUI uses the name of the predefined configuration in the above table.
2. The abbreviation (in Table 6 on page 25) is used with AMDuProfCLI collect command’s --config option.
3. The supported predefined configurations and the sampling events used in them is based on the processor family and model.

2.3 Predefined View Configuration

A View is a set of sampled event data and computed performance metrics either displayed in the GUI or in the text report generated by the CLI. Each predefined sampling configuration has a list of associated predefined views.

Following is the list of predefined view configurations for Assess Performance:

Table 7. Assess Performance Configurations

| View Configuration | Abbreviation | Description |
|--------------------|--------------|-------------|-------------|
The following table lists the predefined view configurations for **Investigate Data Access:**

<table>
<thead>
<tr>
<th>View configuration</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC assessment</td>
<td>ipc_assess</td>
<td>Find hot spots with low instruction level parallelism, provides performance indicators – IPC and CPI.</td>
</tr>
<tr>
<td>Data access assessment</td>
<td>dc_assess</td>
<td>Provides information about data cache (DC) access including DC miss rate and DC miss ratio.</td>
</tr>
<tr>
<td>Misaligned access assessment</td>
<td>misalign_assess</td>
<td>You can use this to identify regions of code that access misaligned data.</td>
</tr>
</tbody>
</table>

The following table lists the predefined view configurations for **Investigate Branch Access:**

<table>
<thead>
<tr>
<th>View configuration</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Investigate Branching</td>
<td>Branch</td>
<td>You can use this view to find code with a high branch density and poorly predicted branches.</td>
</tr>
<tr>
<td>IPC assessment</td>
<td>ipc_assess</td>
<td>Find hotspots with low instruction level parallelism, provides performance indicators – IPC and CPI.</td>
</tr>
<tr>
<td>Branch assessment</td>
<td>br_assess</td>
<td>You can use this view to find code with a high branch density and poorly predicted branches.</td>
</tr>
<tr>
<td>Taken branch report</td>
<td>taken_focus</td>
<td>You can use this view to find the code with a high number of taken branches.</td>
</tr>
</tbody>
</table>
The following table lists the predefined view configurations for **Access Performance (Extended):**

**Table 10. Assess Performance (Extended) Configurations**

<table>
<thead>
<tr>
<th>View configuration</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assess Performance (Extended)</td>
<td>triage_assess_ext</td>
<td>This view gives an overall picture of performance. You can use it to find possible issues for deeper investigation.</td>
</tr>
<tr>
<td>IPC assessment</td>
<td>ipc_assess</td>
<td>Find hotspots with low instruction level parallelism, provides performance indicators – IPC and CPI.</td>
</tr>
<tr>
<td>Branch assessment</td>
<td>br_assess</td>
<td>Use this view to find code with a high branch density and poorly predicted branches.</td>
</tr>
<tr>
<td>Data access assessment</td>
<td>dc_assess</td>
<td>Provides information about data cache (DC) access including DC miss rate and DC miss ratio.</td>
</tr>
<tr>
<td>Misaligned access assessment</td>
<td>misalign_assess</td>
<td>Identify regions of code that access misaligned data.</td>
</tr>
</tbody>
</table>

The following table lists the predefined view configurations for **Investigate Instruction Access:**

**Table 11. Investigate Instruction Access**

<table>
<thead>
<tr>
<th>View configuration</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPC assessment</td>
<td>ipc_assess</td>
<td>Find hotspots with low instruction level parallelism. Provides performance indicators – IPC and CPI.</td>
</tr>
<tr>
<td>Instruction cache report</td>
<td>ic_focus</td>
<td>You can use this view to identify regions of code that miss in the Instruction Cache (IC).</td>
</tr>
<tr>
<td>ITLB report</td>
<td>itlb_focus</td>
<td>You can use this view to analyze and break out ITLB miss rates by levels L1 and L2.</td>
</tr>
</tbody>
</table>

The following table lists the predefined view configurations for **Instruction Based Sampling:**

**Table 12. Instruction Based Sampling Configurations**

<table>
<thead>
<tr>
<th>View configuration</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS fetch overall</td>
<td>ibs_fetch_overall</td>
<td>You can use this view to display an overall summary of the IBS fetch sample data.</td>
</tr>
<tr>
<td>IBS fetch instruction cache</td>
<td>ibs_fetch_ic</td>
<td>You can use this view to display a summary of IBS attempted fetch Instruction Cache (IC) miss data.</td>
</tr>
<tr>
<td>IBS fetch instruction TLB</td>
<td>ibs_fetch_itlb</td>
<td>You can use this view to display a summary of IBS attempted fetch ITLB misses.</td>
</tr>
</tbody>
</table>
Table 12. Instruction Based Sampling Configurations

<table>
<thead>
<tr>
<th>View configuration</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS fetch page translations</td>
<td>ibs_fetch_page</td>
<td>You can use this view to display a summary of the IBS L1 ITLB page translations for attempted fetches.</td>
</tr>
<tr>
<td>IBS All ops</td>
<td>ibs_op_overall</td>
<td>You can use this view to display a summary of all IBS Op samples.</td>
</tr>
<tr>
<td>IBS MEM all load/store</td>
<td>ibs_op_ls</td>
<td>You can use this view to display a summary of IBS Op load/store data.</td>
</tr>
<tr>
<td>IBS MEM data cache</td>
<td>ibs_op_ls_dc</td>
<td>You can use this view to display a summary of DC behavior derived from IBS Op load/store samples.</td>
</tr>
<tr>
<td>IBS MEM data TLB</td>
<td>ibs_op_ls_dtlb</td>
<td>You can use this view to display a summary of DTLB behavior derived from IBS Op load/store data.</td>
</tr>
<tr>
<td>IBS MEM locked ops and access by type</td>
<td>ibs_op_ls_memacc</td>
<td>You can use this view to display the uncachable (UC) memory access, write combining (WC) memory access, and locked load/store operations.</td>
</tr>
<tr>
<td>IBS MEM translations by page size</td>
<td>ibs_op_ls_page</td>
<td>You can use this view to display a summary of DTLB address translations broken out by page size.</td>
</tr>
<tr>
<td>IBS MEM forwarding and bank conflicts</td>
<td>ibs_op_ls_expert</td>
<td>You can use this view to display the memory access bank conflicts, data forwarding, and Missed Address Buffer (MAB) hits.</td>
</tr>
<tr>
<td>IBS BR branch</td>
<td>ibs_op_branch</td>
<td>You can use this view to display the IBS retired branch op measurements including mispredicted and taken branches.</td>
</tr>
<tr>
<td>IBS BR return</td>
<td>ibs_op_return</td>
<td>You can use this view to display the IBS return op measurements including the return misprediction ratio.</td>
</tr>
<tr>
<td>IBS NB local/remote access</td>
<td>ibs_op_nb_access</td>
<td>You can use this view to display the number and latency of local and remote accesses.</td>
</tr>
<tr>
<td>IBS NB cache state</td>
<td>ibs_op_nb_cache</td>
<td>You can use this view to display the cache owned (O) and modified (M) state for NB cache service requests.</td>
</tr>
<tr>
<td>IBS NB request breakdown</td>
<td>ibs_op_nb_service</td>
<td>You can use this view to display the breakdown of NB access requests.</td>
</tr>
</tbody>
</table>

Notes:

1. The AMDuProf GUI uses the **name** of the predefined configuration in the above table.
2. The abbreviation is used with AMDuProfCLI **collect** command’s **--config** option.
3. The supported predefined configurations and the sampling events used in them is based on the processor family and model.
Chapter 3 Getting started with AMDuProfPcm – System Analysis

The System Analysis utility AMDuProfPcm helps to monitor basic performance monitoring metrics for AMD’s family 17h and 19h processors. This utility periodically collects the CPU Core, L3, and DF performance event count values and reports various metrics.

Notes:

1. This tool is supported on Windows, Linux, and FreeBSD.
2. On Linux, AMDuProfPcm uses msr driver and either requires root privileges or read write permissions for /dev/cpu/*/msr devices. NMI watchdog must be disabled (echo 0 > /proc/sys/kernel/nmi_watchdog).
3. On FreeBSD, AMDuProfPcm uses cpuctl module and requires either root privileges or read write permissions for /dev/cpuctl* devices.

Synopsis:

AMDuProfPcm [OPTIONS] -- [PROGRAM] [ARGS]

ARGS — Denotes the launch application to be profiled.

ARGS — Denotes the list of arguments for the launch application.

Common Usages:

$ AMDuProfPcm -h
# AMDuProfPcm -m ipc -c core=0 -d 10 -o /tmp/pmcdata.txt
# AMDuProfPcm -m memory -a -d 10 -o /tmp/memdata.txt -- /tmp/myapp.exe

3.1 Options

The following table lists all the options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Displays this help information on the console/terminal.</td>
</tr>
</tbody>
</table>
The supported metric groups and the corresponding metrics are Platform, OS, and Hypervisor specific. Run `AMDuProfpcm -h` to get the list of supported metrics.

The following metric groups are supported:

- **ipc** – reports metrics such as CEF, Utilization, CPI, and IPC
- **fp** – reports GFLOPS
- **l1** – L1 cache related metrics (DC access and IC Fetch miss ratio)
- **l2** – L2D and L2I cache related access/hit/miss metrics
- **l3** – L3 cache metrics like L3 Access, L3 Miss, and Average Miss latency
- **dc** – advanced caching metrics such as DC refills by source
- **memory** – approximate memory read and write bandwidths in GB/s for all the channels
- **pcie** – PCIe bandwidth in GB/s
- **xgmi** – approximate xGMI outbound databytes in GB/s for all the remote links

Options:

- **-m <metric,...>**
  - Metrics to report, the default metric group is 'ipc'.
  - The supported metric groups and the corresponding metrics are Platform, OS, and Hypervisor specific.
  - Run `AMDuProfpcm -h` to get the list of supported metrics.

- **-c <core|ccx|ccd|package=<n>**
  - Collect from the specified core | ccx | die | package. The default is 'core=0'.
  - If 'ccx' is specified:
    - The core events will be collected from all the cores of this ccx.
    - The l3 and df events will be collected from the first core of this ccx.
  - If 'die' is specified:
    - The core events will be collected from all the cores of this die.
    - The l3 events will be collected from the first core of all the ccx's of this die.
    - The df events will be collected from the first core of this die.
  - If 'package' is specified:
    - The core events will be collected from all the cores of this package.
    - The l3 events will be collected from the first core of all the ccx's of this package.
    - The df events will be collected from the first core of all the die of this package.

- **-a**
  - Collect from all the cores.
  - *Note: Options -c and -a cannot be used together.*

- **-C**
  - Prints the cumulative data at the end of the profile duration. Otherwise, all the samples will be reported as timeseries data.
The following granularities are supported:

- **system** – samples from all the cores in the system will be aggregated
- **package** – samples from all the cores in the package will be aggregated and reported for all the packages available in the system; applicable for multi-package systems.
- **ccd** – samples from all the cores in CCD will be aggregated and reported for all the CCDs.
- **ccx** – samples from all the cores in CCX will be aggregated and reported for all the CCXs.
- **core** – samples from all the cores on which samples are collected will be reported without aggregation.

Notes:
1. Option `-a` should be used along with this option to collect samples from all the cores.
2. Comma separated list of components can be specified.

- **-i <config file>**
  User defined XML config file that specifies Core|L3|DF counters to monitor.
  Refer sample files in `<install-dir>/bin/Data/Config/ dir` for the format.
  Notes:
  1. Options `-i` and `-m` cannot be used together.
  2. If option `-i` is used, all the events mentioned in the user defined config file will be collected.

- **-d <seconds>**
  Profile duration to run.

- **-t <multiplex interval in ms>**
  The interval in which pmc count values will be read, the minimum is 16 ms.

- **-o <output file>**
  The output file name, it is in CSV format.

- **-D <dump file>**
  The output file that contains the event count dump for all the monitored events. It is in CSV format.

- **-p <n>**
  Sets precision of the metrics reported, the default value is 2.

- **-q**
  Hide CPU topology section in the output report.

- **-r**
  Force resets the MSRs.

- **-l**
  Lists the supported raw PMC events.

- **-z <pmc-event>**
  Prints the name, description, and available unit masks for the event.

- **-x <core-id,...>**
  Core affinity for launched application, comma separated list of core IDs.

---

**Table 13. AMDuProfPcm Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| `-A <system,package,ccd,ccx,core>` | Prints aggregated metrics at various component level. The following granularities are supported:  
  - **system** – samples from all the cores in the system will be aggregated  
  - **package** – samples from all the cores in the package will be aggregated and reported for all the packages available in the system; applicable for multi-package systems.  
  - **ccd** – samples from all the cores in CCD will be aggregated and reported for all the CCDs.  
  - **ccx** – samples from all the cores in CCX will be aggregated and reported for all the CCXs.  
  - **core** – samples from all the cores on which samples are collected will be reported without aggregation.  
  Notes:  
  1. Option `-a` should be used along with this option to collect samples from all the cores.  
  2. Comma separated list of components can be specified. |
| `-i <config file>` | User defined XML config file that specifies Core|L3|DF counters to monitor.  
Refer sample files in `<install-dir>/bin/Data/Config/ dir` for the format.  
Notes:  
  1. Options `-i` and `-m` cannot be used together.  
  2. If option `-i` is used, all the events mentioned in the user defined config file will be collected. |
| `-d <seconds>` | Profile duration to run. |
| `-t <multiplex interval in ms>` | The interval in which pmc count values will be read, the minimum is 16 ms. |
| `-o <output file>` | The output file name, it is in CSV format. |
| `-D <dump file>` | The output file that contains the event count dump for all the monitored events. It is in CSV format. |
| `-p <n>` | Sets precision of the metrics reported, the default value is 2. |
| `-q` | Hide CPU topology section in the output report. |
| `-r` | Force resets the MSRs. |
| `-l` | Lists the supported raw PMC events. |
| `-z <pmc-event>` | Prints the name, description, and available unit masks for the event. |
| `-x <core-id,...>` | Core affinity for launched application, comma separated list of core IDs. |
Table 13. AMDuProfPcm Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-w &lt;dir&gt;</td>
<td>Specifies the working directory. The default will be the path of the launched application.</td>
</tr>
<tr>
<td>-v</td>
<td>Print version.</td>
</tr>
</tbody>
</table>

Following are the performance metrics for AMD EPYC™ “Zen 2” core architecture processors:

Table 14. Performance Metrics 1

<table>
<thead>
<tr>
<th>Metric Group</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Utilization (%)</td>
<td>Percentage of time the core was running, that is non-idle time.</td>
</tr>
<tr>
<td></td>
<td>Eff Freq</td>
<td>Core Effective Frequency (CEF) without halted cycles over the sampling period, reported in GHz. The metric is based on APERF and MPERF MSRs. MPERF is incremented by the core at the P0 state frequency while the core is in C0 state. APERF is incremented in proportion to the actual number of core cycles while the core is in C0 state.</td>
</tr>
<tr>
<td>ipc</td>
<td>IPC</td>
<td>Instruction Per Cycle (IPC) is the average number of instructions retired per CPU cycle. This is measured using Core PMC events PMCx0C0 [Retired Instructions] and PMCx076 [CPU Clocks not Halted]. These PMC events are counted in both OS and User mode.</td>
</tr>
<tr>
<td></td>
<td>CPI</td>
<td>Cycles Per Instruction (CPI) is the multiplicative inverse of IPC metric. This is one of the basic performance metrics indicating how cache misses, branch mis-predictions, memory latencies, and other bottlenecks are affecting the execution of an application. A lower CPI value is better.</td>
</tr>
<tr>
<td></td>
<td>Branch Misprediction Ratio</td>
<td>The ratio between mis-predicted branches and retired branch instructions.</td>
</tr>
<tr>
<td>fp</td>
<td>Retired SSE/AVX Flops(GFLOPs)</td>
<td>The number of retired SSE/AVX FLOPs.</td>
</tr>
<tr>
<td></td>
<td>Mixed SSE/AVX Stalls</td>
<td>Mixed SSE/AVX stalls. This metric is in per thousand instructions (PTI).</td>
</tr>
<tr>
<td></td>
<td>IC(32B) Fetch Miss Ratio</td>
<td>Instruction cache fetch miss ratio.</td>
</tr>
<tr>
<td></td>
<td>DC Access</td>
<td>All data cache (DC) accesses. This metric is in PTI.</td>
</tr>
</tbody>
</table>
## Table 14. Performance Metrics 1

<table>
<thead>
<tr>
<th>Metric Group</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2</td>
<td>L2 Access</td>
<td>All the L2 cache accesses. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Access from IC Miss</td>
<td>The L2 cache accesses from IC miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Access from DC Miss</td>
<td>The L2 cache accesses from DC miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Access from HWPF</td>
<td>The L2 cache accesses from L2 hardware pre-fetching. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Miss</td>
<td>All the L2 cache misses. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Miss from IC Miss</td>
<td>The L2 cache misses from IC miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Miss from DC Miss</td>
<td>The L2 cache misses from DC miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Miss from HWPF</td>
<td>The L2 cache misses from L2 hardware pre-fetching. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Hit</td>
<td>All the L2 cache hits. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Hit from IC Miss</td>
<td>The L2 cache hits from IC miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Hit from DC Miss</td>
<td>The L2 cache hits from DC miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Hit from HWPF</td>
<td>The L2 cache hits from L2 hardware pre-fetching. This metric is in PTI.</td>
</tr>
<tr>
<td>tlb</td>
<td>L1 ITLB Miss</td>
<td>The instruction fetches the misses in the L1 Instruction Translation Lookaside Buffer (ITLB), but hit in the L2-ITLB plus the ITLB reloads originating from page table walker. The table walk requests are made for L1-ITLB miss and L2-ITLB misses. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 ITLB Miss</td>
<td>The number of ITLB reloads from page table walker due to L1-ITLB and L2-ITLB misses. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L1 DTLB Miss</td>
<td>The number of L1 Data Translation Lookaside Buffer (DTLB) misses from load store micro-ops. This event counts both L2-DTLB hit and L2-DTLB miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 DTLB Miss</td>
<td>The number of L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops. This metric is in PTI.</td>
</tr>
<tr>
<td>L3</td>
<td>L3 Access</td>
<td>The L3 cache accesses. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L3 Miss</td>
<td>The L3 cache miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>Ave L3 Miss Latency</td>
<td>Average L3 miss latency in core cycles.</td>
</tr>
</tbody>
</table>
Following are the performance metrics for AMD EPYC™ “Zen 2” core architecture processors:

### Table 14. Performance Metrics 1

<table>
<thead>
<tr>
<th>Metric Group</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>Mem Ch-A RdBw (GB/s)</td>
<td>Memory Read and Write bandwidth in GB/s for all the</td>
</tr>
<tr>
<td></td>
<td>Mem Ch-A WrBw (GB/s)</td>
<td>channels.</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>xgmi</td>
<td>xGMI0 BW (GB/s)</td>
<td>Approximate xGMI outbound data bytes in GB/s for all</td>
</tr>
<tr>
<td></td>
<td>xGMI1 BW (GB/s)</td>
<td>the remote links.</td>
</tr>
<tr>
<td></td>
<td>xGMI2 BW (GB/s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xGMI3 BW (GB/s)</td>
<td></td>
</tr>
<tr>
<td>pcie</td>
<td>PCIe0 (GB/s)</td>
<td>Approximate PCIe bandwidth in GB/s.</td>
</tr>
<tr>
<td></td>
<td>PCIe1 (GB/s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCIe2 (GB/s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCIe3 (GB/s)</td>
<td></td>
</tr>
</tbody>
</table>

### Table 15. Performance Metrics 2

<table>
<thead>
<tr>
<th>Metric Group</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilization</td>
<td>Utilization (%)</td>
<td>Percentage of time the core was running, that is non-idle time.</td>
</tr>
<tr>
<td>Eff Freq</td>
<td>Core Effective Frequency (CEF) without halted cycles over the sampling period, reported in GHz. The metric is based on APERF and MPERF MSRs. MPERF is incremented by the core at the P0 state frequency while the core is in C0 state. APERF is incremented in proportion to the actual number of core cycles while the core is in C0 state.</td>
<td></td>
</tr>
<tr>
<td>IPC</td>
<td>Instruction Per Cycle (IPC) is the average number of instructions retired per CPU cycle. This is measured using Core PMC events PMCx0C0 [Retired Instructions] and PMCx076 [CPU Clocks not Halted]. These PMC events are counted in both OS and User mode.</td>
<td></td>
</tr>
<tr>
<td>CPI</td>
<td>Cycles Per Instruction (CPI) is the multiplicative inverse of IPC metric. This is one of the basic performance metrics indicating how cache misses, branch mis-predictions, memory latencies, and other bottlenecks are affecting the execution of an application. A lower CPI value is better.</td>
<td></td>
</tr>
<tr>
<td>Branch Misprediction Ratio</td>
<td>The ratio between mis-predicted branches and retired branch instructions.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 15. Performance Metrics 2

<table>
<thead>
<tr>
<th>Metric Group</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fp</td>
<td>Retired SSE/AVX Flops(GFLOPs)</td>
<td>The number of retired SSE/AVX FLOPs.</td>
</tr>
<tr>
<td></td>
<td>Mixed SSE/AVX Stalls</td>
<td>Mixed SSE/AVX stalls. This metric is in per thousand instructions (PTI).</td>
</tr>
<tr>
<td>1l</td>
<td>IC (32B) Fetch Miss Ratio</td>
<td>Instruction cache fetch miss ratio.</td>
</tr>
<tr>
<td></td>
<td>Op Cache (64B) Fetch Miss Ratio</td>
<td>Operation cache fetch miss ratio.</td>
</tr>
<tr>
<td></td>
<td>IC Access</td>
<td>All instruction cache accesses. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>IC Miss</td>
<td>The instruction cache miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>DC Access</td>
<td>All the DC accesses. This metric is in PTI.</td>
</tr>
<tr>
<td>12</td>
<td>L2 Access</td>
<td>All the L2 cache accesses. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Access from IC Miss</td>
<td>The L2 cache accesses from IC miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Access from DC Miss</td>
<td>The L2 cache accesses from DC miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Access from HWPF</td>
<td>The L2 cache accesses from L2 hardware pre-fetching. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Hit</td>
<td>All the L2 cache hits. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Hit from IC Miss</td>
<td>The L2 cache hits from IC miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Hit from DC Miss</td>
<td>The L2 cache hits from DC miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 Hit from HWPF</td>
<td>The L2 cache hits from L2 hardware pre-fetching. This metric is in PTI.</td>
</tr>
</tbody>
</table>
### Table 15. Performance Metrics 2

<table>
<thead>
<tr>
<th>Metric Group</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tlb</td>
<td>L1 ITLB Miss</td>
<td>The instruction fetches the misses in the L1 Instruction Translation Lookaside Buffer (ITLB), but hit in the L2-ITLB plus the ITLB reloads originating from page table walker. The table walk requests are made for L1-ITLB miss and L2-ITLB misses. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 ITLB Miss</td>
<td>The number of ITLB reloads from page table walker due to L1-ITLB and L2-ITLB misses. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L1 DTLB Miss</td>
<td>The number of L1 Data Translation Lookaside Buffer (DTLB) misses from load store micro-ops. This event counts both L2-DTLB hit and L2-DTLB miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L2 DTLB Miss</td>
<td>The number of L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>All TLBs Flushed</td>
<td>All the TLBs flushed. This metric is in PTI.</td>
</tr>
<tr>
<td>dc</td>
<td>DC Fills from Same CCX</td>
<td>The number of DC fills from local L2 cache to the core or different L2 cache in the same CCX or L3 cache that belongs to the CCX. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>DC Fills from different CCX in same node</td>
<td>The number of DC fills from cache of different CCX in the same package (node). This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>DC Fills from Local Memory</td>
<td>The number of DC fills from DRAM or IO connected in the same package (node). This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>DC Fills from Remote CCX Cache</td>
<td>The number of DC fills from cache of CCX in the different package (node). This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>DC Fills from Remote Memory</td>
<td>The number of DC fills from DRAM or IO connected in the different package (node). This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>All DC Fills</td>
<td>The total number of DC fills from all the data sources. This metric is in PTI.</td>
</tr>
<tr>
<td>l3</td>
<td>L3 Access</td>
<td>The L3 cache accesses. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>L3 Miss</td>
<td>The L3 cache miss. This metric is in PTI.</td>
</tr>
<tr>
<td></td>
<td>Ave L3 Miss Latency</td>
<td>The average L3 miss latency in core cycles.</td>
</tr>
<tr>
<td>Memory</td>
<td>Mem Ch-A RdBw (GB/s)</td>
<td>Memory Read and Write bandwidth in GB/s for all the channels.</td>
</tr>
<tr>
<td></td>
<td>Mem Ch-A WrBw (GB/s)</td>
<td>Memory Read and Write bandwidth in GB/s for all the channels.</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>Memory Read and Write bandwidth in GB/s for all the channels.</td>
</tr>
</tbody>
</table>
### 3.2 Examples

#### 3.2.1 Linux and FreeBSD

- Collect IPC data from core 0 for the duration of 60 seconds:
  ```
  # ./AMDuProfPcm -m ipc -c core=0 -d 60 -o /tmp/pcmdata.csv
  ```
- Collect IPC/L3 metrics for CCX=0 for the duration of 60 seconds:
  ```
  # ./AMDuProfPcm -m ipc,l3 -c ccx=0 -d 60 -o /tmp/pcmdata.csv
  ```
- Collect only the memory bandwidth across all the UMCs for the duration of 60 seconds and save the output in `/tmp/pcmdata.csv` file:
  ```
  # ./AMDuProfPcm -m memory -a -d 60 -o /tmp/pcmdata.csv
  ```
- Collect IPC data for 60 seconds from all the cores:
  ```
  # ./AMDuProfPcm -m ipc -a -d 60 -o /tmp/pcmdata.csv
  ```
- Collect IPC data from core 0 and run the program in core 0:
  ```
  # ./AMDuProfPcm -m ipc -c core=0 -o /tmp/pcmdata.csv -- /usr/bin/taskset -c 0 myapp.exe
  ```
- Collect IPC and data l2 data from core 0 and report the cumulative (not timeseries) and run the program in core 0
  ```
  # ./AMDuProfPcm -m ipc,l2 -c core=0 -o /tmp/pcmdata.csv -C -- /usr/bin/taskset -c 0 myapp.exe
  ```
- List the supported raw Core PMC events:
  ```
  # ./AMDuProfPcm -l
  ```
- Print the name, description, and the available unit masks for the specified event:
  ```
  # ./AMDuProfPcm -z pmcx03
  ```

#### 3.2.2 Windows

**Core Metrics**

- Get the list of supported metrics:
  ```
  C:\> AMDuProfPcm.exe -h
  ```
- Collect IPC data from core 0 for the duration of 30 seconds:
  ```
  C:\> AMDuProfPcm.exe -m ipc -c core=0 -d 30 -o c:\tmp\pcmdata.csv
  ```

---

<table>
<thead>
<tr>
<th>Metric Group</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xgmi</td>
<td>xGMI0 BW (GB/s)</td>
<td>Approximate xGMI outbound data bytes in GB/s for all the remote links.</td>
</tr>
<tr>
<td></td>
<td>xGMI1 BW (GB/s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xGMI2 BW (GB/s)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>xGMI3 BW (GB/s)</td>
<td></td>
</tr>
</tbody>
</table>

---

**Table 15. Performance Metrics 2**
• Collect IPC/L2 metrics for all the core in CCX=0 for the duration of 30 seconds:
  C:\> AMDuProfPcm.exe -m ipc,l2 -c ccx=0 -d 30 -o c:\tmp\pcmdata.csv

• Collect IPC data for 30 seconds from all the cores in the system:
  C:\> AMDuProfPcm.exe -m ipc -a -d 30 -o c:\tmp\pcmdata.csv

• Collect IPC data from core 0 and run the program:
  C:\> AMDuProfPcm.exe -m ipc -c core=0 -o c:\tmp\pcmdata.csv myapp.exe

• Collect IPC and data l2 data from all the cores and report the aggregated data at the system and package level:
  C:\> AMDuProfPcm.exe -m ipc,l2 -a -o c:\tmp\pcmdata.csv -A system,package

• Collect IPC and data l2 data from all the cores in CCX=0 and report the cumulative (not timeseries):
  C:\> AMDuProfPcm.exe -m ipc,l2 -c ccx=0 -o c:\tmp\pcmdata.csv -C -d 30

• Collect IPC and data l2 data from all the cores and report the cumulative (not timeseries):
  C:\> AMDuProfPcm.exe -m ipc,l2 -a -o c:\tmp\pcmdata.csv -C -d 30

• Collect IPC and data l2 data from all the cores and report the cumulative (not timeseries) and aggregate at system and package level:
  C:\> AMDuProfPcm.exe -m ipc,l2 -a -o c:\tmp\pcmdata.csv -C -A system,package -d 30

L3 Metrics

• Collect L3 data from ccx=0 for the duration of 30 seconds:
  C:\> AMDuProfPcm.exe -m l3 -c ccx=0 -d 30 -o c:\tmp\pcmdata.csv

• Collect L3 data from all the CCXs and report for the duration of 30 seconds:
  C:\> AMDuProfPcm.exe -m l3 -a -d 30 -o c:\tmp\pcmdata.csv

• Collect L3 data from all the CCXs and aggregate at system and package level and report for the duration of 30 seconds:
  C:\> AMDuProfPcm.exe -m l3 -a -d 30 -A system,package -o c:\tmp\pcmdata.csv

• Collect L3 data from all the CCXs and aggregate at system and package level and report for the duration of 30 seconds; also report for the individual CCXs:
  C:\> AMDuProfPcm.exe -m l3 -a -d 30 -A system,package,ccx -o c:\tmp\pcmdata.csv

• Collect L3 data from all the CCXs for the duration of 30 seconds and report the cumulative data (no timeseries data):
  C:\> AMDuProfPcm.exe -m l3 -a -d 30 -C -o c:\tmp\pcmdata.csv

• Collect L3 data from all the CCXs and aggregate at system and package level and report cumulative data (no timeseries data)
  C:\> AMDuProfPcm.exe -m l3 -a -d 30 -A system,package -C -o c:\tmp\pcmdata.csv

• Collect IPC data from core 0 for the duration of 30 seconds:
  C:\> AMDuProfPcm.exe -m ipc -c core=0 -d 30 -o c:\tmp\pcmdata.csv
Memory Bandwidth

- Report memory bandwidth for all the memory channels for the duration of 60 seconds and save the output in `c:\tmp\pcmdata.csv` file:

  ```
  C:\> AMDuProfPcm.exe -m memory -a -d 60 -o c:\tmp\pcmdata.csv
  ```

- Report total memory bandwidth aggregated at the system level for the duration of 60 seconds and save the output in `c:\tmp\pcmdata.csv` file:

  ```
  C:\> AMDuProfPcm.exe -m memory -a -d 60 -o c:\tmp\pcmdata.csv -A system
  ```

- Report total memory bandwidth aggregated at the system level and also report for every memory channel:

  ```
  C:\> AMDuProfPcm.exe -m memory -a -d 60 -o c:\tmp\pcmdata.csv -A system,package
  ```

- Report total memory bandwidth aggregated at the system level and also report for all the available memory channels. To report cumulative metric value instead of the timeseries data:

  ```
  C:\> AMDuProfPcm.exe -m memory -a -d 60 -o c:\tmp\pcmdata.csv -C -A system,package
  ```

Raw Event Count Dump

- Monitor events from core 0 and dump the raw event counts for every sample in timeseries manner, no metrics report will be generated:

  ```
  C:\> AMDuProfPcm.exe -m ipc -d 60 -D c:\tmp\pcmdata_dump.csv
  ```

- Monitor events from all the cores and dump the raw event counts for every sample in timeseries manner, no metrics report will be generated:

  ```
  C:\> AMDuProfPcm.exe -m ipc -a -d 60 -D c:\tmp\pcmdata_dump.csv
  ```

Custom Config File

A sample config XM file is available in `<uprof-install-dir>\bin\Data\Config\SamplePcm-core.conf`. This file can be copied and modified to certain user-specific interesting events and formula to compute metrics. All the metrics defined in that file will be monitored and reported.

```
C:\> AMDuProfPcm.exe -i SamplePcm-core.conf -a -d 60 -o c:\tmp\pcmdata.csv
C:\> AMDuProfPcm.exe -i SamplePcm-core-l3-df.conf -a -d 60 -o c:\tmp\pcmdata.csv
```  

Miscellaneous

- List the supported raw Core PMC events:

  ```
  C:\> AMDuProfPcm.exe -1
  ```

- Print the name, description, and the available unit masks for the specified event:

  ```
  C:\> AMDuProfPcm.exe -z pmcx03
  ```
AMDuProfSys is a python-based system analysis tool for AMD processors. In Linux, this tool is based on the underlaying Linux perf utility provided by the Linux kernel, msr module, and basic hardware access primitives provided by the kernel. In Windows, AMDuProfSys uses the underlaying driver to collect the PMU event counters. This tool can be used to collect the hardware events and evaluate the simple counter values or complex recipe using collected raw events.

**Supported Platforms**
- Family 17, model 0x30 - 0x3F
- Family 19, model 0x0 - 0xF

**Supported Hardware Counters**
- CORE PMC
- DF PMC
- L3 PMC

**Supported Operating Systems**
- Linux
- Windows (Only core PMC metrics are supported)

The AMDuProfSys tool is available in the following installation directory:

<Installed Directory>/bin/AMDPerf/AMDuProfSys.py

**Prerequisites**

In Linux, user space tool for Linux perf module should be installed for core, l3, and df hardware counter access. If perf is not installed already, you can install it using following command on Ubuntu:

```
$ sudo apt-get install linux-tools-common linux-tools-generic linux-tools-`uname -r`
```

In Linux, NMI watchdog must be disabled, this requires root privileges:

```
$ sudo echo 0 > /proc/sys/kernel/nmi_watchdog
```

By default, tool expects Python 3.6. To run the tool in python 2, the command must be prefixed with python2. The following python modules must be installed:

- **tqdm** — use pip3 install tqdm to install
- **xlsxwriter** — use pip3 install XlsxWriter to install
- **yaml** — use apt-get install python-yaml or pip3 install pyyaml to install
- **yamlOrderedDictLoader** — use pip3 install yamlOrderedDictLoader to install
Synopsis

AMDuProfSys [<OPTIONS>] -- [<PROGRAM>] [<ARGS>]

<OPTIONS> — To collect, generate report, or get help for this tool

<PROGRAM> — Denotes a launch application to be profiled

<ARGS> — Denotes the list of arguments for the launch application

Common Usages

- Display help:
  
  ```
  $ ./:AMDuProfSys -h
  ```

- Collect core metrics for core 0 and run application on core:
  
  ```
  $./AMDuProfSys.py collect --config core -C 0 -o output taskset -c 0 <application>
  ```

- Generate the .csv format report from the session file generated during collection:
  
  ```
  $./AMDuProfSys.py report -i output_core.ses
  ```

- Generate report in .xls format:
  
  ```
  $./AMDuProfSys.py report -i output_core.ses -f xls
  ```

- Collect time series core metrics, at an interval of 1000 ms for core 0-5 and run application on core:
  
  ```
  $./AMDuProfSys.py collect --config core -C 0-5 -I 1000 -o output taskset -c 0 <application>
  ```

- Generate time series report in .csv format:
  
  ```
  $./AMDuProfSys.py report -i output_core.ses
  ```

- Collect metrics for core, L3, and DF metrics together:
  
  ```
  #./AMDuProfSys.py collect --config core,l3,df,umc -C 0-10 <application>
  ```

4.1 Options

4.1.1 Collect Command

The following table lists the collect command options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>Displays this help information on the console/terminal.</td>
</tr>
<tr>
<td>--config</td>
<td>To launch the given application and to monitor the raw events. Collect commands can be configured to use predefined set of config files or a single config file with its path.</td>
</tr>
<tr>
<td>-a</td>
<td>Collect from all the cores.</td>
</tr>
</tbody>
</table>
  
  *Note: Options -c and -a cannot be used together.*
Table 16. AMDuProfSys Collect Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>List of CPUs to monitor. Multiple CPUs can be provided as comma separated list with no space: 0,1. Ranges of CPUs: 5-10.</td>
</tr>
<tr>
<td>-d &lt;seconds&gt;</td>
<td>Profile duration to run.</td>
</tr>
<tr>
<td>-t, --tid &lt;tid&gt;</td>
<td>Monitor events on existing thread(s). Multiple TIDs can be provided.</td>
</tr>
<tr>
<td>-p, --pid &lt;pid&gt;</td>
<td>Monitor events on existing process(es). Multiple PIDs can be provided as comma separated list.</td>
</tr>
<tr>
<td>-o &lt;output folder&gt;</td>
<td>Output raw data path. This option is with collect command</td>
</tr>
<tr>
<td>-I, --interval &lt;n&gt;</td>
<td>Interval at which raw event count deltas will be stored in the file. This is a must for collecting time series data.</td>
</tr>
<tr>
<td>-V</td>
<td>Print verbose.</td>
</tr>
</tbody>
</table>

Table 17. AMDuProfSys Report Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>report</td>
<td>To generate the profile report with computed metrics. Previous collect command generates a profile session file with .ses extension and a the raw counter data file for each type of profile collection. To generate the report, you must provide the session file with a -i option as shown in the command options.</td>
</tr>
<tr>
<td>-i, --input-file &lt;file&gt;</td>
<td>Input the session file generated by collect command. This option has to be provided with report option.</td>
</tr>
<tr>
<td>-o, --output &lt; folder&gt;</td>
<td>Output file name in .csv or .xls format as configured.</td>
</tr>
<tr>
<td>-f, --format</td>
<td>Output file format in .xls or .csv. Default file format is .csv.</td>
</tr>
<tr>
<td>--group-by &lt;system,package,numa,ccx&gt;</td>
<td>Aggregate result based on group selected. Default is none.</td>
</tr>
<tr>
<td>-T, --time-series</td>
<td>Generate per core time series report. Only .csv format is supported. Must be collected with -I option to generate the time series data.</td>
</tr>
<tr>
<td>-V</td>
<td>Print verbose.</td>
</tr>
</tbody>
</table>

4.1.2 Report Command

The following table lists the report command options:

Creating Customized Config Profile

The AMDuProfSys profile uses the following three type of input files for profiling:

- Event Files

Event files are in data\0xXX_YY\events\folder. XX and YY are family and model of the supported AMD processors. These files define the respective hardware PMC counters with their unit mask.
and other respective attributes. These events are used in the metrics defined in the metrics file. Event files are in .json format and begin with the prefix of the counter type. For example, core related event file should begin with core_.

- Metrics Files
  Metrics files are in data\0xXX_YY\metrics\ folder. XX and YY are family and model of the supported AMD processors. Metrics are the recipe or formula for a meaningful parameter. For example, IPC is a metrics calculated by the formula:

  \[ \text{ipc} = \frac{(\text{retired\_instructions})}{((\text{tsc})*(\text{aperf})/(\text{mperf}))} \]

  Events used in the metrics must be declared in the corresponding event file. Metrics can be as simple as the raw pmc count (example, retired\_instructions) or a complex metrics as IPC. Further, metrics can also be defined as hierarchy of another metrics. Metrics files are in .json format and begin with the prefix of the counter type. For example, core related event file should begin with core_.

- Config Files
  Config files are in data\0xXX_YY\configs\ folder. XX and YY are family and model of the supported AMD processors. The config file has three sections:
  - **CPU** consists of the supported processors, various PMC hardware counters availability.
  - **Events** are declared for configuring raw PMC events in each multiplexed group. Since, HW PMC counters are limited, multiplexing group must be configured.
  - **Reports** generation using the metrics defined in the metrics file. Based on the list of metrics configured, report will be generated. Ensure that the required events for evaluating metrics are configured for collection in the event section. The config files are in yaml format and begin with the prefix of the counter type. For example, core related event file must begin with core_.

Chapter 4  Getting Started with AMDuProfSys – System Analysis  44
Chapter 5 Getting Started with AMD uProf GUI

5.1 User Interface

The AMD uProf GUI provides a visual interface to profile and analyze the performance data. It has various pages and each page has several sub-windows. You can navigate the pages through the top horizontal navigation bar. When a page is selected, its sub-windows will be listed in the leftmost vertical pane as follows:

1. The menu names in the horizontal bar such as HOME, PROFILE, SUMMARY, and ANALYZE are called pages.

2. Each page has its sub-windows listed in the leftmost vertical pane. For example, HOME page has various windows such as Welcome, Recent Session(s), Import Session, and so on.

3. Each window has various sections. These sections are used to specify various inputs required for a profile run, display the profile data for analysis, buttons and links to navigate to associated sections. In the Welcome window, Quick Links section has two links that allows you to start a profile session with minimal configuration steps.
5.2 Launching GUI

To launch the AMDuProf GUI program:

Windows
Launch GUI from `C:\Program Files\AMD\AMDuProf\bin\AMDuProf.exe` or using the Desktop shortcut.

Linux
Launch GUI from `/opt/AMDuProf_X.Y-ZZZ/AMDuProf` binary.

The Welcome screen is displayed as follows:

![AMD uProf Welcome Screen](image)

Figure 2. AMD uProf Welcome Screen

It has many sections as follows:

1. **Start Here** section provides quick links to start profile for the various profile targets.
2. Recently used profile configurations are listed in **Recently Used Configuration(s)** section. You can click on this configuration to reuse that profile configuration for subsequent profiling.
3. Recently opened profile sessions are listed in **Recently Opened Session(s)** section. You can click on any one of the sessions to load the corresponding profile data for further analysis.
4. **Quick Links** section contains two entries which lets you to start profiles with minimal configuration.
a. Click **See what’s keeping your System busy** to start a system-wide time-based profiling until you stop it and then display the collected data.

b. Click **See what’s guzzling power in your System** to select various power and thermal related counters and display a live view of the data through graphs.

5. **AMD uProf Resources** section provides links to the AMD uProf release page and AMD server community forum for discussions on profiling and performance tuning.

### 5.3 Configure a Profile

To perform a collect run, first you should configure the profile by specifying the:

1. **Profile target**

2. **Profile type**
   a. What profile data should be collected (CPU Profile or Live Power Profile)
   b. Monitoring events - how the data should be collected
   c. Additional profile data (if needed) - callstack samples, profile scheduling, and so on

This is called profile configuration “Profile Configuration” on page 24 that identifies all the information used to perform a collect measurement.

*Note:* The additional profile data to be collected depends on the selected profile type.
5.3.1 Select Profile Target

To start a profile, either click the PROFILE page at the top navigation bar or Profile an Application? link in HOME page Welcome screen. The Start Profiling screen is displayed. Select Profile Target is available in the Start Profiling window as follows:

Figure 3. Start Profiling - Select Profile Target

You can select the one of the following profile targets from the Select Profile Target drop-down:

- **Application**: Select this target when you want to launch an application and profile it (or launch and do a system-wide profile). The only compulsory option is a valid path to the executable. (By default, the path to the executable becomes the working directory unless you specify a path).
- **System**: Select this if you do not wish to launch any application but perform either a system-wide profile or profile specific set of cores.
- **Process(es)**: Select this if you want to profile an application/process which is already running. This will bring up a process table which can be refreshed. Selecting any one of the processes from the table is mandatory to start profile.

Once profile target is selected and configured with valid data, the **Next** button will be enabled to go the next screen of Start Profiling.

**Note:** The **Next** button will be enabled only if all the selected options are valid.
5.3.2 Select Profile Type

Once profile target is selected and configured, click the Next button. The Select Profile Type screen is displayed as follows:

![Select Profile Type Screen]

This screen lets you decide the type of profile data collected and how the data should be collected. You can select the profile type based on the performance analysis that you intend to perform. In the above figure:

1. **Select Profile Type** drop-down lists all the supported profile types.
2. Once you select a profile type, the left vertical pane within this window will list the options corresponding to the selected profile type. For **CPU Profile** type, all the available predefined sampling configurations will be listed.
3. This section lists all the sampling events that are monitored in the selected predefined sampling configuration. Each entry represents a sampling configuration (unit mask, sampling interval, OS, and user mode) for that event. You can click **Modify Events** button to modify these event attributes and add new events and/or remove events.
4. Click **Advanced Options** button to proceed to the **Advanced Options** screen and set the other options such as the **Call Stack Options**, **Profile Scheduling**, **Sources**, **Symbols**, and so on.
5. The details in “Profile Configuration” on page 24 are persistent and saved by the tool with a name (here, it is **AMDuProf-EBP-ScimarkStable**). You can define this name and navigate to **PROFILE > Saved Configurations** to reuse/select the same configuration later.
6. The **Next** and **Previous** buttons are available to navigate to various screen of the **Start Profiling** screen.

### 5.3.3 Advanced Options

Click the **Advanced Options** button in **Select Profile Type** screen. The **Advanced Options** screen is displayed as follows:

![Advanced Options Screen](image)

**Figure 5.** **Start Profiling - Advanced Options 1**
You can set the following options on the Advanced Options screen:

1. **OS Tracing Options** to collect the profile data to show Thread Concurrency Chart and other OS events. The supported tracing events are OS specific.

2. **Call Stack Options** to enable callstack sample data collection. This profile data is used to show Flame Graph and Call Graph views.

3. **Profile Scheduling** to schedule the profile data collection.

4. The **Next** and **Previous** buttons are available to navigate to various fragments within the Start Profiling screen.

5. **Sources** line-edit to specify the path(s) to locate the source files of the profiled application.

6. **Symbols** to specify the Symbols servers (Windows only) and to specify the path(s) to locate the symbol files of the profiled application.
5.3.4 Start Profile

Once all the options are set correctly, click the Start Profile button to start the profile and collect the profile data. After the profile initialization the following screen is displayed:

![Figure 7. Profile Data Collection](image)

1. The time elapsed during the data collection is displayed.
2. When the profiling is in progress, you can:
   - Click the Stop button to stop the profiling.
   - Clicking Cancel button to cancel the profiling. It will take you back to Select Profile Target screen of PROFILE.
   - Click the Pause button to pause the profiling. The profile data will not be collected and you can click the Resume button to continue the profiling.

5.4 Analyze the Profile Data

When the profiling stopped, the collected raw profile data will be processed automatically and you can analyze the profile data through various UI sections to identify the potential performance bottlenecks:

- SUMMARY page to look at overview of the hotspots for the profile session.
- ANALYZE page to examine the profile data at various granularities.
- SOURCES page to examine the data at source line and assembly level.
- MEMORY page to examine the cache-line data for potential false cache sharing.
- HPC page to examine the OpenMP tracing data for potential load imbalance issue.
The sections available depend on the profile type. The CPU Profile will have SUMMARY, ANALYZE, MEMORY, HPC, and SOURCES pages to analyze the data.

5.4.1 Overview of Performance Hotspots

When the translation is complete, the SUMMARY page will be populated with the profile data and Hot Spots screen will be displayed. The SUMMARY page provides an overview of the hotspots for the profile session through various screens such as Hot Spots and Session Information.

In the Hot Spots screen, hotspots will be displayed for functions, modules, process, and threads. Processes and threads will be displayed only if there are more than one.

The following figure shows the Hot Spots screen:

![Figure 8. Summary - Hot Spots Screen](image)

In the above Hot Spots screen:

1. The top 5 hottest functions, processes, modules and threads for the selected event are displayed.
2. The Hot Functions pie chart is interactive in nature. You can click on any section and the corresponding function's source will open in a separate tab in the SOURCES page.

The hotspots are shown per event and the monitored event can be selected from drop-down in the top-right corner. You can change it to any other event to update the corresponding hotspot data.
5.4.2 Thread Concurrency Graph

Click **ANALYZE > Thread Concurrency** to view the following graph to analyze the thread concurrency of the profiled application:

![Thread Concurrency Graph](image)

**Figure 9. Summary - Thread Concurrency Graph**
5.4.3 Function HotSpots

Click the ANALYZE button on the top horizontal navigation bar to goto Function HotSpots screen, which displays the hot functions across all the profiled processes and load modules as follows:

Figure 10. ANALYZE - Function HotSpots

Function HotSpots screen contains the following:

1. The Functions table lists the hot functions. The IP samples are aggregated and attributed at the function-level granularity. On the table, you can do the following:
   - Double-click on a function entry to navigate to the corresponding SOURCE view of that function.
   - Right-click to view the following options:
     - Copy selected rows(s) to copy the highlighted row to clipboard.
     - Copy all rows to copy all the rows to clipboard.

2. Filters and Options pane allows you filter the profile data as follows:
   - You can click the Select View drop-down to control the counters that are displayed. The relevant counters and their derived metrics are grouped in predefined views.
   - You can use the Value Type drop-down to display the counter values as follows:
     - Sample Count is the number of samples attributed to a function.
     - Event Count is the product of sample count and sampling interval.
– **Percentage** is the percentage of samples collected for a function.

- You can use the **System Modules** option to either **Exclude** or **Include** the profile data attributed to system modules.

3. If callstack is enabled, the unique hot call-paths for the selected function is displayed in the **Functions** column.

4. **Event Timeline** is the line graph showing the number of aggregated sample values over the period of time. You can use it to identify the hot functions within a profile region. From the Select Metric drop-down you can select the event for which event timeline must be plotted.

   All the entries will not be loaded for a profile. To load more than the default number of entries, click the vertical scroll bar on the right.

### 5.4.4 Process and Functions

Click **ANALYZE > Metrics** to display the profile data table at various program unit granularities - Process, Load Modules, Threads, and Functions. This screen contains data in two different formats as follows:

![Analyse - Metrics](image)

**Figure 11. Analyze - Metrics**
The above figure consists of the following:

1. The upper tree represents samples grouped by **Process**. You can expand the tree to view the child entries for each parent (that is for a process). The **Load Modules** and **Threads** are child entries for the selected process entry.

   You can right-click to view the following options:
   - **Expand All Entries** to list the modules and threads of all the processes.
   - **Collapse All Entries** to list only the top-level entries.
   - **Copy selected row(s)** to copy the highlighted row to clipboard.
   - **Copy all rows** to copy all the rows to clipboard.

2. The lower **Functions** table contains samples attributed to corresponding functions. The function entries depend on what is selected in the upper tree. For more specific data, you can select a child entry from the upper tree and the corresponding function data will be updated in the lower tree.

   You can do any of the following:
   - Double-click on a function entry to navigate to the corresponding SOURCE view.
   - Right-click to view the following options:
     - **Copy selected row(s)** to copy the highlighted row to clipboard.
     - **Copy all rows** to copy all the rows to clipboard.

3. You can use the **Filters and Options** pane to filter the profile data displayed by various controls.

   - The **View** controls the counters that are displayed. The relevant counters and their derived metrics are grouped in predefined views. You can select the views from the **View** drop-down.
   - The **Group By** drop-down is used to group the data by Process, Module, and Thread. By default, the sample data is grouped-by Process.
   - Click the **Show Values By** drop-down to display the counter values as follows:
     - **Sample Count** is the number of samples attributed to a function.
     - **Event Count** is the product of sample count and sampling interval.
     - **Percentage** is the percentage of samples collected for a function.
   - You can use the **System Modules** option to **Exclude** or **Include** the profile data attributed to system modules.

4. Confidence level — The metrics that cannot be calculated reliably due to low number of samples collected for a program unit will be grayed out.

   All entries will not be loaded for a profile. To load more than the default number of entries, click the vertical scroll bar on the right.
## 5.4.5 Source and Assembly

Double-click on any entry in the **Functions** table in the **Metrics** screen to load the source tab for the corresponding function in **SOURCES** page. If the GUI can find the path to the source file for that function, then it will try to open the file, failing which you will be prompted to locate it.

The following figure depicts the source and assembly screen:

![Source and Assembly Screen](image)

**Figure 12. SOURCES - Source and Assembly**

Following section are present in the **SOURCES** screen:

1. The source lines of the selected function are listed and the corresponding metrics are populated in various columns against each source line. If no samples are collected when a source line was executed, the metrics column will be empty.

2. The assembly instruction of the corresponding highlighted source line. The tree will also show the offset for each assembly instruction along with metrics.

3. Heatmap – overview of the hotspots at source level.
4. **Filters** pane lets you filter the profile data by providing the following options.

   - The **View** controls the counters that are displayed. The relevant counters and their derived metrics are grouped in predefined views. You can select it from the **View** drop-down.
   - The **PID** drop-down lists all the processes on which this selected function is executed and has samples.
   - The **TID** drop-down lists all the threads on which this selected function is executed and has samples.
   - You can use the **Show Values By** drop-down to display the counter values as follows:
     - **Sample Count** is the number of samples attributed to a function.
     - **Event Count** is the product of sample count and sampling interval.
     - **Percentage** is the percentage of samples collected for a function.

For multi-threaded or multi-process applications, if a function has been executed from multiple threads or processes, each of them will be listed in the **PID** and **TID** drop-downs in the **Filters** pane. Changing them will update the profile data for that selection. By default, profile data for the selected function, aggregated across all processes and all threads will be displayed.

**Note:** If the source file cannot be located or opened, only disassembly will be displayed.

### 5.4.6 Flame Graph

Flame graph is a visualization of sampled call-stack traces to quickly identify the hottest code execution paths. Click **ANALYZE > Flame Graph** to view it as follows:

![Flame Graph](image)

**Figure 13.** ANALYZE - Flame Graph
The **Flame Graph** screen comprises of the following:

1. The x-axis of the flame graph shows the call-stack profile and the y-axis shows the stack depth. It is not plotted based on passage of time. Each cell represents a stack frame and if a frame were present more often in the call-stack samples, the cell would be wider. This screen has the following options:
   - Module-wise coloring of the cells.
   - Click on a cell to zoom only that cell and its children. Use the **Reset Zoom** button visualize the entire graph.
   - Right-click on a cell to view the following context options:
     - **Copy Function Data** to copy the function names and its metrics to clipboard.
     - **Open Source View** to navigate to the source tab of that function.
   - Hover the mouse over a cell to display the tool-tip showing the inclusive and exclusive number of samples of that function.

2. Following options are available at the top of this screen:
   - Click **Zoom Graph** button for a better zooming experience.
   - Searching for particular function will highlight that function cells in all the flame graph.
   - The **Process** drop-down lists all the processes for which call-stack samples are collected. Changing the process will plot the flame graph for that particular process.
   - For multi-threaded applications, the flame graph will be plotted for the hottest thread by default.
   - The **Threads** drop-down lists all the threads for which call-stack samples are collected. Changing the thread will plot the flame graph for that thread.
   - The **Select Metric** drop-down lists all the metrics for which call-stack samples are collected. Changing the metric will plot the flame graph for that particular metric.
5.4.7 Call Graph

Click ANALYZE > Call Graph to navigate to the call graph screen. This graph is constructed using the call-stack samples and offers a butterfly view to analyze the hot call-paths as follows:

1. The Function table lists all the functions with inclusive and exclusive samples. Click on function to display its Caller and Callee functions in a butterfly view.
2. Lists all the parents of the function selected in the Function table.
3. Lists all the children of the function selected in the Function table.
4. Options:
   - The Process drop-down lists all the processes for which call-stack samples are collected. Changing the process will show the call graph for that particular process.
   - For multi-threaded applications, the call-graph will be plotted for the hottest thread by default.
   - The Threads drop-down lists all the threads for which call-stack samples are collected. Changing the thread will plot the call graph for that thread.
   - The Select Metric drop-down lists all the counters for which call-stack samples are collected. Changing the counter will show the call graph for that particular counter.

![Call Graph Screen](image)
5.5 Importing Profile Database

To analyze a profile database generated using CLI, click HOME > Import Session to go to the Import Profile Session. The following screen is displayed:

![Import Profile Session Screen](image)

Figure 15. Import Session – Importing Profile Database

This can be used to import the processed profile data collected using the CLI or the processed profile data saved in GUI’s profile session storage path. You must do the following:

- Specify the path in the Profile Data File box.
- **Binary Path**: If the profile run is performed in a system and the corresponding raw profile data is imported in another system, you must specify the path(s) in which binary files can be located.
- **Source Path**: Specify the source path(s) from where the sources files can be located.
- **Symbols**: Specify the symbol path(s) from where the debug info files (On Windows, PDB files) can be located.
5.6 Analyzing Saved Profile Session

Once you have created a new profile session or opened (imported) profile database, the history is updated and the last 50 opened profile database records are stored (that is, where they are located). Such a list will also appear in HOME > Recent Session(s) as follows:

![Recent Session(s) figure]

Figure 16. PROFILE - Recent Session(s)

In the above figure:

1. History of profile sessions opened for analysis in the GUI. The following options are available:
   - Click on an entry to load the corresponding profile database for analysis.
   - See Details button displays details about this profile session such as profiled application, monitored events list, and so on.
   - Click Edit Options to automatically fill the Import Profile Session for the database and update the required line-edits before opening the session.
   - Remove Entry button deletes the current profile session from the history.
2. Displays the details of the selected profile session.
5.7 Using Saved Profile Configuration

When a profile configuration is created (when you set the options and start profiling), if it generates at least one valid profile session, the profile configuration details will be stored with the options set and can be loaded again. Such a list is available in PROFILE > Saved Configurations as follows:

![PROFILE - Saved Configurations](image)

Figure 17. PROFILE - Saved Configurations

In the above figure:

1. History of profile configurations used to collect profile data using GUI. The following options are available:
   - Click on an entry to display the corresponding profile configuration for data collection.
   - **See Details** button displays the details about the current profile session such profiled application, monitored events list, and so on.
   - **Remove Entry** button deletes the current profile session from the history.

2. Displays the details of the selected profile session.

**Note:** By default, the profile configuration name is generated by the application. If you want to reuse it, you should name it appropriately to locate it easily. To do so, provide a config name in the bottom left corner (**Config Name** line-edit) in PROFILE > **Start Profiling**.
5.8 Settings

There are certain application-wide settings to customize the AMD uProf experience. The SETTINGS page is in top-right corner and is divided into the following three sections:

- **Preferences**: Use this section to set the global path and data reporting preferences

![SETTINGS - Preferences](image)

**Figure 18. SETTINGS - Preferences**

- Click the **Apply Changes** button to apply the updated/modified settings. There are settings which are common to profile data filters and hence, any changes to them through the **Apply Changes** button will only be applied to the views that do not have local filters set.
- You can click **Reset** button to reset the settings or **Cancel** to discard the changes that you don't want to apply.
• **Symbols**: Use this section to configure the Symbol Paths and Symbol Server locations. The Symbol server is a Windows only option. The following figure represents the **Symbols** section:

![Figure 19. SETTINGS - Symbols](image)

• **Source Data**: Use this section to set the Source view preferences. The following figure represents the **Source Data** section:

![Figure 20. SETTINGS - Source Data](image)
Chapter 6  Getting Started with CLI

AMD uProf’s command line interface AMDuProfCLI provides options to collect and generate report for analyzing the profile data.

`AMDuProfCLI [--version] [--help] COMMAND [options] [PROGRAM] [ARGS]`

The following commands are supported:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collect</td>
<td>Runs the given program and collects the profile samples.</td>
</tr>
<tr>
<td>report</td>
<td>Processes the raw profile datafile and generates profile report.</td>
</tr>
<tr>
<td>timechart</td>
<td>Power Profiling — collects and reports system characteristics, such as power, thermal, and frequency metrics.</td>
</tr>
<tr>
<td>info</td>
<td>Displays the generic information about system and topology.</td>
</tr>
<tr>
<td>translate</td>
<td>Processes the raw profile datafile and generates the profile DB.</td>
</tr>
</tbody>
</table>

For more information on the workflow, refer “Workflow and Key Concepts” on page 23. To run the command line interface AMDuProfCLI run the following binaries as per the OS:

- **Windows**
  
  C:\Program Files\AMD\AMDuProf\bin\AMDuProfCLI.exe

- **Linux**:
  
  /opt/AMDuProf_X.Y-ZZZ/AMDuProfCLI
  
  or
  
  ./AMDuProf_Linux_x64_X.Y.ZZZ/bin/AMDuProfCLI (if installed using the .tar file)

- **FreeBSD**:
  
  ./AMDuProf_FreeBSD_x64_X.Y.ZZZ/bin/AMDuProfCLI

### 6.1  Starting a CPU Profile

To profile and analyze the performance of a native (C/C++) application, you must complete the following steps:

1. Prepare the application. For more information on preparing an application for profiling, refer “Reference” on page 163.
2. Use AMDuProfCLI’s `collect` command to collect the samples for the application.
3. Using AMDuProfCLI’s `report` command to generate a report in readable format for analysis.
Preparing the application is to build the launch application with debug information as it is needed to correlate the samples to functions and source lines.

The `collect` command launches the application (if given) and collects the profile data for the given profile type and sampling configuration. It generates the raw data file (.prd on Windows and .caperf on Linux) and other miscellaneous files.

The `report` command translates the collected raw profile data to aggregate and attribute to the respective processes, threads, load modules, functions, and instructions. Also, it writes them into a database and then generates a report in the CSV file format.

The following figure shows how to run time-based profile and generate a report for the launch application `AMDTClassicMatMul.exe`:

![Figure 21. Collect and Report Commands](image)

### 6.1.1 List of Predefined Sample Configurations

To get the list of supported predefined sampling configurations that can be used with `collect` command’s `--config` option, run the following command:

```
C:\> AMDuProfCLI.exe info --list collect-configs
```
A sample output is as follows:

```
C:\Program Files\AMD\AMDuProf\bin>AMDuProfCLI.exe info --list collect-configs
list of predefined profiles that can be used with 'collect --config' option:

  tbp  : Time-based Sampling
       Use this configuration to identify where programs are spending time.

  inst_access : Investigate Instruction Access
       Use this configuration to find instruction fetches with poor L1 instruction
       cache locality and poor ITLB behavior.
       [PMU Events: PMC0,076, PMC0,080, PMC0,088, PMC0,081, PMC0,084, PMC0,085]

  energy : Power Profile
       Use this configuration to identify where programs are consuming power.

  data_access : Investigate Data Access
       Use this configuration to find data access operations with poor L1 data
       cache locality and poor DTLB behavior.
       [PMU Events: PMC0,076, PMC0,080, PMC0,089, PMC0,086, PMC0,083, PMC0,084, PMC0,085]

  branch : Investigate Branching
       Use this configuration to find poorly predicted branches and near returns.
       [PMU Events: PMC0,076, PMC0,080, PMC0,089, PMC0,086, PMC0,083, PMC0,084, PMC0,085]

  assess_ext : Assess Performance (Extended)
       This configuration has additional events to monitor than the Assess Performance
       configuration. Use this configuration to get an overall assessment of performance.
       [PMU Events: PMC0,076, PMC0,080, PMC0,089, PMC0,086, PMC0,083, PMC0,084, PMC0,085]

  assess : Assess Performance
       Use this configuration to get an overall assessment of performance and
       to find potential issues for investigation.
       [PMU Events: PMC0,080, PMC0,076, PMC0,089, PMC0,086, PMC0,083, PMC0,084, PMC0,085]
```

Figure 22. List Supported Predefined Configurations

### 6.1.2 Profile Report

The profile report (in CSV format) contains the following sections:

- **EXECUTION** — Information about the target launch application.
- **PROFILE DETAILS** — Details about the current session, such as profile type, scope, and sampling events.
- **5 HOTTEST Functions** — List of the top 5 hot functions and the metrics attributed to them.
• PROFILE REPORT FOR PROCESS — The metrics attributed to the profiled process. This section contains other sub-sections such as:
  – THREAD SUMMARY — List of threads with metrics attributed to them.
  – MODULE SUMMARY — List of load modules which belong to the process with metrics attributed to them.
  – FUNCTION SUMMARY — List of functions that belong to this process for which samples are collected, with metrics attributed to them.
  – Function Detail Data — Source level attribution for the top functions for which samples are collected.
  – CALLGRAPH — Call graph, if callstack samples are collected.

6.2 Starting a Power Profile

6.2.1 System-wide Power Profiling (Live)

To collect power profile counter values, complete the following steps:

1. Run the AMDuProfCLI’s `timechart` command with `--list` option to get the list of supported counter categories.

2. Use the AMDuProfCLI’s `timechart` command for specifying the required counters with `--event` option to collect and the report the required counters.

The `timechart` run to list the supported counter categories:

![Figure 23. Output of timechart --list Command](image)

Figure 23. Output of timechart --list Command
The timechart to collect the profile samples and write into a file:

![Image of timechart execution](image)

**Figure 24. Execution of timechart**

The above run collects the power and frequency counters on all the devices on which these counters are supported and writes them in the output file specified with `-o` option. Before the profiling begins, the given application is launched and the data is collected till the application terminates.

### 6.3 Collect Command

The `collect` command runs the given program, collects the performance profile data, and writes into the specified raw profile data file. This file can then be analyzed using AMDuProfCLI’s `report` command or AMDuProf GUI.

**Synopsis:**

```
AMDuProfCLI collect [--help] [<options>] [<PROGRAM>] [<ARGS>]
```

- `<PROGRAM>` — Denotes the launch application to be profiled.
- `<ARGS>` — Denotes the list of arguments for the launch application.

**Common Usages:**

```
$ AMDuProfCLI collect <PROGRAM> [<ARGS>]
$ AMDuProfCLI collect [--config <config> | -e <event>] [-a] [-d <duration>] [<PROGRAM>]
```

#### 6.3.1 Options

The following table lists the collect command options:

**Table 19. AMDuProfCLI Collect Command Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-h</code></td>
<td><code>--help</code></td>
</tr>
<tr>
<td><code>-o</code>, <code>--output-dir</code></td>
<td><code>&lt;directory-path&gt;</code></td>
</tr>
<tr>
<td><code>--config &lt;config&gt;</code></td>
<td>Predefined sampling configuration to be used to collect samples. Use the command <code>info --list collect-configs</code> to get the list of supported configs. Multiple occurrences of <code>--config</code> are allowed.</td>
</tr>
</tbody>
</table>
-e, --event <EVENT> or <predefined-event>

A predefined event can be directly be used with -e, --event which has predefined arguments.
Use the command info --list predefined-events for the list of supported predefined events.
Alternatively, for providing more granular parameters, specify Timer, PMU, IBS event, or a predefined event with arguments in the form of comma separated key=value pairs. The supported keys are:
- event=<timer | ibs-fetch | ibs-op> or <PMU-event>
- umask=<unit-mask>
- user=<0 | 1>
- os=<0 | 1>
- interval=<sampling-interval>
- ibsop-count-control=<0 | 1>

Note: It is not required to provide umask with predefined event.

Use the option call-graph to collect the callgraph for an event.
Use command info --list predefined-events for the list of supported predefined events.
Use command info --list pmu-events for the list of supported PMU-events.

When these arguments are not passed, then the default values are:
- umask=0
- user=1
- os=1
- ibsop-count-control=0
- interval=1.0 ms for timer event
- interval=250000 for <ibs-fetch | ibs-op> or <PMU-event>

Multiple occurrences of --event (-e) are allowed.

-p | --pid <PID...>
Profile the existing processes (to attach to). The process IDs are separated by comma.

-a | --system-wide
System Wide Profile (SWP)
If this flag is not set, then the command line tool will profile only the launched application or the Process IDs attached with -p option.

-c | --cpu <core...>
Comma separated list of CPUs to profile. The ranges of CPUs can be specified with '-', for example, 0-3.

Note: On Windows, the selected cores should belong to only one processor group. For example, 0-63, 64-127, and so on.

Table 19. AMDuProfCLI Collect Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-e, --event &lt;EVENT&gt;</td>
<td>A predefined event can be directly be used with -e, --event which has</td>
</tr>
<tr>
<td>or &lt;predefined-event&gt;</td>
<td>predefined arguments.</td>
</tr>
<tr>
<td></td>
<td>Use the command info --list predefined-events for the list of supported</td>
</tr>
<tr>
<td></td>
<td>predefined events.</td>
</tr>
<tr>
<td></td>
<td>Alternatively, for providing more granular parameters, specify Timer, PMU,</td>
</tr>
<tr>
<td></td>
<td>IBS event, or a predefined event with arguments in the form of comma</td>
</tr>
<tr>
<td></td>
<td>separated key=value pairs. The supported keys are:</td>
</tr>
<tr>
<td></td>
<td>- event=&lt;timer</td>
</tr>
<tr>
<td></td>
<td>- umask=&lt;unit-mask&gt;</td>
</tr>
<tr>
<td></td>
<td>- user=&lt;0</td>
</tr>
<tr>
<td></td>
<td>- os=&lt;0</td>
</tr>
<tr>
<td></td>
<td>- interval=&lt;sampling-interval&gt;</td>
</tr>
<tr>
<td></td>
<td>- ibsop-count-control=&lt;0</td>
</tr>
<tr>
<td></td>
<td>Use the option call-graph to collect the callgraph for an event.</td>
</tr>
<tr>
<td></td>
<td>Use command info --list predefined-events for the list of supported</td>
</tr>
<tr>
<td></td>
<td>predefined events.</td>
</tr>
<tr>
<td></td>
<td>Use command info --list pmu-events for the list of supported PMU-events.</td>
</tr>
<tr>
<td></td>
<td>When these arguments are not passed, then the default values are:</td>
</tr>
<tr>
<td></td>
<td>- umask=0</td>
</tr>
<tr>
<td></td>
<td>- user=1</td>
</tr>
<tr>
<td></td>
<td>- os=1</td>
</tr>
<tr>
<td></td>
<td>- ibsop-count-control=0</td>
</tr>
<tr>
<td></td>
<td>- interval=1.0 ms for timer event</td>
</tr>
<tr>
<td></td>
<td>- interval=250000 for &lt;ibs-fetch</td>
</tr>
<tr>
<td></td>
<td>Multiple occurrences of --event (-e) are allowed.</td>
</tr>
<tr>
<td>-p</td>
<td>--pid &lt;PID...&gt;</td>
</tr>
<tr>
<td></td>
<td>separated by comma.</td>
</tr>
<tr>
<td>-a</td>
<td>--system-wide</td>
</tr>
<tr>
<td></td>
<td>If this flag is not set, then the command line tool will profile only the</td>
</tr>
<tr>
<td></td>
<td>launched application or the Process IDs attached with -p option.</td>
</tr>
<tr>
<td>-c</td>
<td>--cpu &lt;core...&gt;</td>
</tr>
<tr>
<td></td>
<td>specified with '-', for example, 0-3.</td>
</tr>
<tr>
<td></td>
<td>Note: On Windows, the selected cores should belong to only one processor</td>
</tr>
<tr>
<td></td>
<td>group. For example, 0-63, 64-127, and so on.</td>
</tr>
</tbody>
</table>
### 6.3.2 Windows Specific Options

The following table lists Linux specific collect commands:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| **--call-graph <I:D:S:F>**  | Enables Callstack Sampling. Specify the Unwind Interval (I) in milliseconds and Unwind Depth (D) value. Specify the Scope (S) by choosing one of the following:  
  • user: Collect only for the user space code.  
  • kernel: Collect only for the kernel space code.  
  • all: Collect for the code executed in the user and kernel space code.  
  Specify to collect missing frames due to Frame Pointer Omission (F) by compiler:  
  • fpo: Collect missing callstack frames.  
  • nofpo: Ignore missing callstack frames. |
| **--no-inherit**            | Do not profile the children of the launched application (processes launched by the profiled application).                                    |
| **-b | --terminate**           | Terminate the launched application after the profile data collection ends. Only the launched application process will be killed. Its children (if any) may continue to execute. |
| **--affinity <core...>**   | Set the core affinity of the launched application to be profiled. Comma separated list of core-ids. The ranges of the core-ids must be specified, for example, 0-3. The default affinity is all the available cores. |
| **--start-delay <n>**      | Start delay n in seconds. Start profiling after the specified duration. When n is 0, there is no impact.                                         |
| **--start-paused**         | Profiling paused indefinitely. The target application resumes the profiling using the profile control APIs. This option must be used only when the launched application is instrumented to control the profile data collection using the resume and pause APIs (defined in the AMDProfileControl library). |
| **-v | --verbose <n>**          | Specify debug log messaging level. Valid values of (n) are: 1: INFO, 2: DEBUG, 3: EXTENSIVE                                                                 |
| **--interval <num>**       | Sampling interval for PMC events.                                                                                                           |
| **--duration <n>**         | Profile only for the specified duration n in seconds.                                                                                      |

---

**Table 19. AMDuProfCLI Collect Command Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-g</strong></td>
<td>Same as passing <strong>--call-graph 1:128:user:nofpo</strong>.</td>
</tr>
</tbody>
</table>

**Table 20. Collect Command – Windows Specific Options**
### 6.3.3 Linux Specific Options

The following table lists Linux specific collect commands:

**Table 21. Collect Command – Linux Specific Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| --call-graph <F:N> | Enables Callstack sampling. Specify (F) to collect/ignore missing frames due to omission of frame pointers by compiler:  
  - fpo: Collect missing callstack frames.  
  - nofpo: Ignore missing callstack frames.  
  When F = fpo, (N) specifies the max stack-size in bytes to collect per sample collection. Valid range of the stack size: 16 - 8192. If (N) is not multiple of 8, then it is aligned down to the nearest value multiple of 8. The default value is 1024 bytes.  
  *Note: Passing a large N value will generate a very large raw data file.*  
  When F = nofpo, the value for N is ignored and hence, there is no need to pass it. |
| -g | Same as passing --call-graph nofpo |
| --tid <TID,...> | Profile existing threads (to attach to). The thread IDs are separated by comma. |
| --trace <TARGET> | To trace a target domain. TARGET can be one or more of the following:  
  - mpi[=<lwt|full>] — 'lwt' for light-weight tracing. 'full' for complete tracing. Default is 'full' tracing.  
  - openmp — for tracing OpenMP application. This is same as the option --omp.  
  - os[=<event1,event2,...>] — provide event names and optional threshold with comma separated list. syscall, memtrace, and pthread events can take an optional threshold value as <event:threshold>. If no event specified, then schedule, syscall, and pthread are enabled in launch application or attached PID. memtrace and pthread are not supported with system wide tracing. Thread concurrency information is captured when the schedule event is specified. Use the command info --list ostrace-events for the list of supported OS trace events.  
  - gpu[=<hip,hsa,kfd>] — provide the domain for GPU Tracing. By default, the domain is set to 'hip,hsa,kfd'. |
Table 21. Collect Command – Linux Specific Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--buffer-size &lt;size&gt;</td>
<td>Number of pages to be allotted for OS trace buffer. Default value is 256 pages per core. Increase the pages to reduce the trace data loss. This option is only applicable to OS tracing (--trace os).</td>
</tr>
<tr>
<td>-m, --mmap-pages &lt;size&gt;</td>
<td>Set the kernel memory mapped data buffer to size. The size can be specified in pages or with a suffix Bytes (B/b), Kilo bytes (K/k), Megabytes (M/m), and Gigabytes (G/g).</td>
</tr>
<tr>
<td>--omp</td>
<td>Profile OpenMP application.</td>
</tr>
<tr>
<td></td>
<td>Notes:</td>
</tr>
<tr>
<td></td>
<td>1. Applicable to process profiling.</td>
</tr>
<tr>
<td></td>
<td>2. Not applicable to System wide and Java app profiling.</td>
</tr>
<tr>
<td></td>
<td>3. Compile the OpenMP application with LLVM/Clang 8.0 or later. The supported base languages are C, C++, and Fortran.</td>
</tr>
<tr>
<td>--mpi</td>
<td>Use this option to get the MPI profiling information. For MPI tracing, check the --trace option.</td>
</tr>
<tr>
<td>--thread &lt;thread=concurrency&gt;</td>
<td>Thread concurrency</td>
</tr>
</tbody>
</table>

6.3.4 Examples

Windows

- Launch application `AMDTClassicMatMul.exe` and collect samples for CYCLES_NOT_IN_HALT and RETired_INST events:

  C:\> AMDuProfCLI.exe collect -e cycles-not-in-halt -e retired-inst --interval 1000000  
  -o c:\Temp\cpuprof-custom AMDTClassicMatMul.exe

  C:\> AMDuProfCLI.exe collect -e event=cycles-not-in-halt,interval=250000  
  -e event=retired-inst,interval=500000 -o c:\Temp\cpuprof-custom AMDTClassicMatMul.exe

- Launch the application `AMDTClassicMatMul.exe` and collect Time-Based Profile (TBP) samples:

  C:\> AMDuProfCLI.exe collect -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe

- Launch `AMDTClassicMatMul.exe` and do Assess Performance profile for 10 seconds:

  C:\> AMDuProfCLI.exe collect --config assess -o c:\Temp\cpuprof-assess -d 10 AMDTClassicMatMul.exe

- Launch `AMDTClassicMatMul.exe` and collect IBS samples in the SWP mode:

  C:\> AMDuProfCLI.exe collect --config ibs -a -o c:\Temp\cpuprof-ibs-swp AMDTClassicMatMul.exe

- Collect the TBP samples in SWP mode for 10 seconds:

  C:\> AMDuProfCLI.exe collect -a -o c:\Temp\cpuprof-tbp-swp -d 10

- Launch `AMDTClassicMatMul.exe` and collect TBP with callstack sampling:

  C:\> AMDuProfCLI.exe collect --config tbp -g -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe
• Launch AMDTClassicMatMul.exe and collect TBP with callstack sampling (unwind FPO optimized stack):

```
C:\> AMDuProfCLI.exe collect --config tbp --call-graph 1:64:user:fpo -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe
```

• Launch AMDTClassicMatMul.exe and collect samples for PMCx076 and PMCx0C0:

```
C:\> AMDuProfCLI.exe collect -e event=pmcx76,interval=250000 -e event=pmcxc0,user=1,os=0,interval=250000 -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe
```

• Launch AMDTClassicMatMul.exe and collect samples for IBS OP with interval 50000:

```
C:\> AMDuProfCLI.exe collect -e event=ibs-op,interval=50000 -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe
```

• Launch AMDTClassicMatMul.exe and do TBP samples profile for thread concurrency, name:

```
C:\> AMDuProfCLI.exe collect --config tbp --thread thread=concurrency,name -o c:\Temp\cpuprof-tbp AMDTClassicMatMul.exe
```

• Launch AMDTClassicMatMul.exe and collect Power samples in SWP mode:

```
C:\> AMDuProfCLI.exe collect --config power -a -o c:\Temp\pwrprof-swp AMDTClassicMatMul.exe
```

• Collect samples for PMCx076 and PMCx0C0, but collect call graph info only for PMCx0C0:

```
C:\> AMDuProfCLI collect -e event=pmcx76,interval=250000 -e event=pmcxc0,interval=250000,call-graph -o c:\Temp\cpuprof-pmc AMDTClassicMatMul-bin
```

• Launch AMDTClassicMatMul.exe and collect samples for predefined event RETIRED_INST and L1_DC_ACCESSES.ALL events:

```
C:\> AMDuProfCLI.exe collect -e event=RETIRED_INST,interval=250000 -e event=L1_DC_ACCESSES.ALL,user=1,os=0,interval=250000 -o c:\Temp\cpuprof-pmc AMDTClassicMatMul.exe
```

• Launch AMDTClassicMatMul.exe, collect TBP and Assess Performance samples:

```
C:\> AMDuProfCLI.exe collect --config tbp --config assess -o c:\Temp\cpuprof-tbp-assess AMDTClassicMatMul.exe
```

**Linux**

• Launch application AMDTClassicMatMul.exe and collect samples for CYCLES_NOT_IN_HALT and RETIRED_INST events:

```
$ ./AMDuProfCLI collect -e cycles-not-in-halt -e retired-inst --interval 1000000 -o /tmp(cpuprof-custom AMDTClassicMatMul-bin
C:\> AMDuProfCLI.exe collect -e event=cycles-not-in-halt,interval=250000 -e event=retired-inst,interval=500000 -o /tmp\cpuprof-custom AMDTClassicMatMul-bin
```

• Launch the application AMDTClassicMatMul-bin and collect the TBP samples:

```
$ ./AMDuProfCLI collect -o /tmp\cpuprof-tbp AMDTClassicMatMul-bin
```

• Launch AMDTClassicMatMul-bin and do Assess Performance profile for 10 seconds:

```
$ ./AMDuProfCLI collect --config assess -o /tmp\cpuprof-assess -d 10 AMDTClassicMatMul-bin
```
• Launch AMDTClassicMatMul-bin and collect IBS samples in the SWP mode:

```bash
$ ./AMDuProfCLI collect --config ibs -a -o /tmp/cpuprof-ibs-swp AMDTClassicMatMul-bin
```

• Collect the TBP samples in SWP mode for 10 seconds:

```bash
$ ./AMDuProfCLI collect -a -o /tmp/cpuprof-tbp-swp -d 10
```

• Launch AMDTClassicMatMul-bin and collect TBP with callstack sampling:

```bash
$ ./AMDuProfCLI collect --config tbp -g -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin
```

• Launch AMDTClassicMatMul-bin and collect TBP with callstack sampling (unwind FPO optimized stack):

```bash
$ ./AMDuProfCLI collect --config tbp --call-graph fpo:512 -o /tmp/uprof-tbp AMDTClassicMatMul-bin
```

• Launch AMDTClassicMatMul-bin and collect samples for PMCx076 and PMCx0C0:

```bash
$ ./AMDuProfCLI collect -e event=pmcx76,interval=250000 -e event=pmcx0c0,user=1,os=0,interval=250000 -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin
```

• Launch AMDTClassicMatMul-bin and collect samples for IBS OP with interval 50000:

```bash
$ ./AMDuProfCLI collect -e event=ibs-op,interval=50000 -o /tmp/cpuprof-tbp AMDTClassicMatMul-bin
```

• Attach to a thread and collect TBP samples for 10 seconds:

```bash
$ AMDuProfCLI collect --config tbp -o /tmp/cpuprof-tbp-attach -d 10 --tid <TID>
```

• Collect OpenMP trace info of an OpenMP application, pass --omp:

```bash
$ AMDuProfCLI collect --omp --config tbp <path-to-openmp-exe>
```

• Launch `AMDTClassicMatMul-bin` and collect memory accesses for false cache sharing:

```bash
$ AMDuProfCLI collect --config memory -o /tmp/cpuprof-mem AMDTClassicMatMul-bin
```

• Collect MPI profiling information:

```bash
$ mpirun -np 4 ./AMDuProfCLI collect --config assess --mpi --output-dir /tmp/cpuprof-mpi /tmp/namd <parameters>
```

• Collect samples for PMCx076 and PMCx0C0, but collect call graph info only for PMCx0C0:

```bash
$ AMDuProfCLI collect -e event=pmcx76,interval=250000 -e event=pmcx0c0,interval=250000,call-graph -o /tmp/cpuprof-pmc AMDTClassicMatMul-bin
```

• Launch `AMDTClassicMatMul-bin` and collect samples for predefined event RETIRED_INST and L1_DC_ACCESS.ALL events:

```bash
$ AMDuProfCLI collect -e event=RETIRED_INST,interval=250000 -e event=L1_DC_ACCESS.ALL,user=1,os=0,interval=250000 -o /tmp/cpuprof-pmc AMDTClassicMatMul-bin
```

• Launch `AMDTClassicMatMul-bin` and collect all the OS trace events:

```bash
$ AMDuProfCLI collect --trace os -o /tmp/cpuprof-os AMDTClassicMatMul-bin
```
• Launch `AMDTClassicMatMul-bin` and collect syscall taking more than or equal to 1ms:
  
  ```bash
  $ AMDuProfCLI collect --trace os=syscall:1000 -o /tmp/cpuprof-os AMDTClassicMatMul-bin
  ```

• Launch `AMDTClassicMatMul-bin` and collect the GPU Traces for hip domain:
  
  ```bash
  $ AMDuProfCLI collect --trace gpu=hip -o /tmp/cpuprof-gpu AMDTClassicMatMul-bin
  ```

• Launch `AMDTClassicMatMul-bin` and collect the GPU Traces for hip, hsa, and kfd domain:
  
  ```bash
  $ AMDuProfCLI collect --trace gpu -o /tmp/cpuprof-gpu AMDTClassicMatMul-bin
  ```

• Launch `AMDTClassicMatMul-bin`, collect TBP samples and GPU Traces for hip domain:
  
  ```bash
  $ AMDuProfCLI collect --config tbp --trace gpu=hip -o /tmp/cpuprof-gpu AMDTClassicMatMul-bin
  ```

• Launch `AMDTClassicMatMul-bin` and collect GPU samples:
  
  ```bash
  $ AMDuProfCLI collect --config gpu -o /tmp/cpuprof-gpu AMDTClassicMatMul-bin
  ```

• Launch `AMDTClassicMatMul-bin`, collect GPU samples and OS Traces:
  
  ```bash
  $ AMDuProfCLI collect --config gpu --trace os -o /tmp/cpuprof-gpu-os AMDTClassicMatMul-bin
  ```

• Launch `AMDTClassicMatMul-bin`, collect TBP and GPU samples:
  
  ```bash
  $ AMDuProfCLI collect --config gpu --config tbp -o /tmp/cpuprof-gpu-tbp AMDTClassicMatMul-bin
  ```

### 6.4 Report Command

The report command processes the raw profile data or the processed file and generates a profile report. The profile report can also be generated from the database file.

**Synopsis:**

```
AMDuProfCLI report [--help] [<options>]
```

**Common Usages:**

$ AMDuProfCLI report -i <session-dir path>
### 6.4.1 Options

#### Table 22. AMDuProfCLI Report Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`-h</td>
<td>--help`</td>
</tr>
<tr>
<td><code>-i, --input-dir &lt;directory-path&gt;</code></td>
<td>Path to the directory containing collected data.</td>
</tr>
<tr>
<td><code>--detail</code></td>
<td>Generate detailed report.</td>
</tr>
<tr>
<td><code>--group-by &lt;section&gt;</code></td>
<td>Specify the report to be generated. The supported report options are:</td>
</tr>
<tr>
<td></td>
<td>• process: Report process details</td>
</tr>
<tr>
<td></td>
<td>• module: Report module details</td>
</tr>
<tr>
<td></td>
<td>• thread: Report thread details</td>
</tr>
<tr>
<td></td>
<td>This option is applicable only with <code>--detail</code> option. The default is group-by process.</td>
</tr>
<tr>
<td><code>-p, --pid &lt;PID,..&gt;</code></td>
<td>Generate report for the specified PIDs. The process IDs are separated by comma.</td>
</tr>
<tr>
<td><code>-g</code></td>
<td>The print callgraph. Use with options <code>--detail</code> or <code>--pid(-p)</code> With <code>--pid</code> option, callgraph will be generated only if the callstack samples were collected for specified PIDs.</td>
</tr>
<tr>
<td><code>--cutoff &lt;n&gt;</code></td>
<td>Cutoff to limit the number of process, threads, modules, and functions to be reported. n is the minimum number of entries to be reported in various report sections. The default value is 10.</td>
</tr>
<tr>
<td><code>--view &lt;config&gt;</code></td>
<td>Report only the events present in the given view file. Use the command <code>info --list view-configs</code> to get the list of supported view-configs.</td>
</tr>
<tr>
<td><code>--inline</code></td>
<td>Show inline functions for C, C++ executables.</td>
</tr>
<tr>
<td></td>
<td><em>Note: Using this option will increase the time taken to generate the report.</em></td>
</tr>
<tr>
<td><code>--show-sys-src</code></td>
<td>Generate detailed function report of the system module functions (if debug info is available) with the source statements.</td>
</tr>
<tr>
<td><code>--src-path &lt;path1;...&gt;</code></td>
<td>Source file directories (semicolon separated paths). Multiple use of <code>--src-path</code> is allowed.</td>
</tr>
<tr>
<td><code>--disasm</code></td>
<td>Generate a detailed function report with assembly instructions.</td>
</tr>
<tr>
<td><code>--disasm-only</code></td>
<td>Generate the function report with only assembly instructions.</td>
</tr>
</tbody>
</table>
Table 22. AMDuProfCLI Report Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| -s | --sort-by <EVENT>     | Specify the Timer, PMC, or IBS event on which the reported profile data will be sorted with arguments in the form of comma separated key=value pairs. The supported keys are:  
|                          | • event=<timer | ibs-fetch | ibs-op | pmcxNNN> where NNN is hexadecimal Core PMC event id.  
|                          | • umask=<unit-mask>                                                        |  
|                          | • user=<0 | 1>                                                              |  
|                          | • os=<0 | 1>                                                              |  
|                          | Use the command info --list pmu-events for the list of supported PMC events.  
|                          | Details about the arguments:  
|                          | • umask — Unit mask in decimal or hexadecimal. Applicable only to the PMC events.  
|                          | • user, os — User and OS mode. Applicable only to the PMC events.  
|                          | Multiple occurrences of --sort-by (-s) are not allowed.                  |
| --agg-interval <INTERVAL> | Accumulation of samples for the given INTERVAL in milliseconds. Default INTERVAL is 1 second. If the interval value set to 0 or negative, granularity of the data aggregation would be the complete profile duration. |
| --time-filter <T1:T2>     | Restricts report generation to the time interval between T1 and T2. Where, T1 and T2 are time in seconds from profile start time. |
| --agg-interval <INTERVAL> | Accumulation of samples for the given INTERVAL in milliseconds. Default INTERVAL is 1 millisecond. |
| --imix                   | Generate instruction MIX report.                                           |
| --ignore-system-module   | Ignore samples from system modules.                                       |
| --show-percentage        | Show percentage of samples instead of actual samples.                     |
| --show-sample-count      | Show the number of samples. This option is enabled by default.            |
| --show-event-count       | Show the number of events occurred                                       |
| --bin-path <path>        | Binary file path, multiple usage of --bin-path is allowed.                |
| --symbol-path <path1;...> | Debug Symbol paths (semicolon separated). Multiple use of --symbol-path is allowed. |
| -v | --verbose <n>         | Specify debug log messaging level. The valid values are:  
|                          | 1.: INFO                                                                  |
|                          | 2.: DEBUG                                                                 |
|                          | 3.: EXTENSIVE                                                             |
6.4.2 Windows Specific Options

Table 23. AMDuProfCLI Report Command Windows Specific Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--symbol-server &lt;path1;...&gt;</td>
<td>Symbol Server directories (semicolon separated paths). For example, Microsoft Symbol Server (<a href="https://msdl.microsoft.com/download/symbols">https://msdl.microsoft.com/download/symbols</a>). Multiple use of --symbol-server is allowed.</td>
</tr>
<tr>
<td>--symbol-cache-dir &lt;path&gt;</td>
<td>The path to store the symbol files downloaded from the Symbol Servers.</td>
</tr>
</tbody>
</table>

6.4.3 Linux Specific Options

Table 24. AMDuProfCLI Report Command Linux Specific Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
</table>
| --host <hostname>  | This option is used along with the --input-dir option. Generates report belonging to a specific host. The supported options are:  
  • <hostname>: Report process belonging to a specific host.  
  • all: Report all the processes.  
  **Note:** If --host is not used, only the processes belonging to the system from which report is generated is reported. In case, the system is a master node in a cluster, the report will be generated for the lexicographically first host in that cluster. |
| --category <PROFILE> | Generate report only for specific profiling category. Comma separated multiple categories can be specified. If this option is not used, then report for all categories gets generated. Multiple instance of --category is allowed.  
  Supported categories are:  
  • cpu – Generate report specific to CPU Profiling.  
  • mpi – Generate report specific to MPI Tracing.  
  • openmp – Generate report specific to OpenMP Tracing.  
  • os – Generate report specific to OS Tracing.  
  • gputrace – Generate report specific to GPU Tracing.  
  • gpuprof – Generate report specific to GPU Profiling.  
  Example:  
  --category cpu,mpi,os,gputrace,gpuprof  
  --category mpi --category cpu --category os --category gputrace --category gpuprof |
| --limit-cacheinfo <n> | Cut-off limit for the entries in the cache line analysis report sections. The default value is 10.                                                                                                           |
6.4.4 Examples

Windows

- Generate report from the raw datafile:

  ```
  C:\> AMDuProfCLI.exe report -i c:\Temp\cpuprof-tbp\<SESSION-DIR>
  ```

- Generate IMIX report from the raw datafile:

  ```
  C:\> AMDuProfCLI.exe report --imix -i c:\Temp\cpuprof-tbp\<SESSION-DIR>
  ```

- Generate report from the raw datafile sorted on pmc event:

  ```
  C:\> AMDuProfCLI.exe report -s event=pmcxc0,user=1,os=0 -i c:\Temp\cpuprof-ebp\<SESSION-DIR>
  ```

- Generate report from the raw datafile sorted on ibs-op event:

  ```
  C:\> AMDuProfCLI.exe report -s event=ibs-op -i c:\Temp\cpuprof-ibs\<SESSION-DIR>
  ```

- Generate report from the raw datafile for power samples:

  ```
  C:\> AMDuProfCLI.exe report -i c:\Temp\pwrprof-swp\<SESSION-DIR>
  ```

- Generate report with Symbol Server paths:

  ```
  C:\> AMDuProfCLI.exe report --symbol-path C:\AppSymbols;C:\DriverSymbols --symbol-server http://msdl.microsoft.com/download/symbols --symbol-cache-dir C:\symbols -i c:\Temp\cpuprof-tbp\<SESSION-DIR>
  ```

- Generate report from the raw datafile on one of the predefined views:

  ```
  C:\> AMDuProfCLI.exe report --view ipc_assess -i c:\Temp\pwrprof-swp\<SESSION-DIR>
  ```

- Generate report from the raw datafile providing the source and binary paths:

  ```
  C:\> AMDuProfCLI.exe report --bin-path Examples\AMDTClassicMatMul\bin\ --src-path Examples\AMDTClassicMatMul\ -i c:\Temp\cpuprof-tbp\<SESSION-DIR>
  ```

Linux

- Generate report from the raw datafile:

  ```
  $ AMDuProfCLI report -i /tmp/cpuprof-tbp/<SESSION-DIR>
  ```

- Generate IMIX report from the raw datafile:

  ```
  $ AMDuProfCLI report --imix -i /tmp/cpuprof-tbp/<SESSION-DIR>
  ```

- Generate report from the raw datafile sorted on pmc event:

  ```
  $ AMDuProfCLI report -s event=pmcxc0,user=1,os=0 -i /tmp/cpuprof-ebp/<SESSION-DIR>
  ```

- Generate report from the raw datafile sorted on ibs-op event:

  ```
  $ AMDuProfCLI report -s event=ibs-op -i /tmp/cpuprof-ibs/<SESSION-DIR>
  ```

- Generate OS Trace report from the raw datafile:

  ```
  $ AMDuProfCLI report -i /tmp/cpuprof-os/<SESSION-DIR> --category os
  ```
6.5 Timechart Command

This timechart command collects and reports the system characteristics, such as power, thermal and frequency metrics, and generates a text or CSV report.

Synopsis:

```
AMDuProfCLI timechart [--help] [list] [<options>] [<PROGRAM>] [<ARGS>]
```

- `<PROGRAM>` — Denotes the application to be launched before starting the power metrics collection.
- `<ARGS>` — Denotes the list of arguments for the launch application.

Common Usages:

```
$ AMDuProfCLI timechart --list
$ AMDuProfCLI timechart -e <event> -d <duration> [<PROGRAM>] [<ARGS>]
```
6.5.1 Options

Table 25. Timechart Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--help</td>
</tr>
<tr>
<td>--list</td>
<td>Displays all the supported devices and categories.</td>
</tr>
<tr>
<td>-e</td>
<td>--event &lt;type...&gt;</td>
</tr>
<tr>
<td>-t</td>
<td>--interval &lt;n&gt;</td>
</tr>
<tr>
<td>-d</td>
<td>--duration &lt;n&gt;</td>
</tr>
<tr>
<td>--affinity &lt;core...&gt;</td>
<td>The core affinity. Comma separated list of core-ids. Ranges of core-ids is also be specified, for example, 0-3. The default affinity is all the available cores. The affinity is set for the launched application.</td>
</tr>
<tr>
<td>-w</td>
<td>--working-dir &lt;dir&gt;</td>
</tr>
<tr>
<td>-f</td>
<td>--format &lt;fmt&gt;</td>
</tr>
<tr>
<td></td>
<td>• txt: Text (.txt) format.</td>
</tr>
<tr>
<td></td>
<td>• csv: Comma Separated Value (.csv) format.</td>
</tr>
<tr>
<td>-o</td>
<td>--output-dir &lt;dir&gt;</td>
</tr>
</tbody>
</table>

6.5.2 Examples

Windows

- Collect all the power counter values for a duration of 10 seconds with sampling interval of 100 milliseconds:

  ```
  C:\> AMDuProfCLI.exe timechart --event power --interval 100 --duration 10
  ```

- Collect all the frequency counter values for 10 seconds, sampling them every 500 milliseconds and dumping the results into a `csv` file:

  ```
  C:\> AMDuProfCLI.exe timechart --event frequency -o C:\Temp\output --interval 500 --duration 10
  ```

- Collect all the frequency counter values at core 0 to 3 for 10 seconds, sampling them every 500 milliseconds and dumping the results into a text file:

  ```
  C:\> AMDuProfCLI.exe timechart --event core=0-3,frequency -o C:\Temp\PowerOutput --interval 500 -duration 10 --format txt
  ```
Linux

- Collect all the power counter values for a duration of 10 seconds with sampling interval of 100 milliseconds:
  
  ```
  $ ./AMDuProfCLI timechart --event power --interval 100 --duration 10
  ```
- Collect all the frequency counter values for 10 seconds, sampling them every 500 milliseconds and dumping the results into a csv file:
  
  ```
  $ ./AMDuProfCLI timechart --event frequency -o /tmp/PowerOutput --interval 500 --duration 10
  ```
- Collect all the frequency counter values at core 0 to 3 for 10 seconds, sampling them every 500 milliseconds and dumping the results into a text file:
  
  ```
  $ ./AMDuProfCLI timechart --event core=0-3,frequency -o /tmp/PowerOutput --interval 500 --duration 10 --format txt
  ```

### 6.6 Info Command

This command gets the generic information about the system, CPU topology, disassembly of a binary, and so on.

**Synopsis:**

```
AMDuProfCLI info [--help] [options]
```

**Common Usages:**

```
$ AMDuProfCLI info --system
```
### 6.6.1 Options

#### Table 26. Info Command Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-h</td>
<td>--help</td>
</tr>
<tr>
<td>--list &lt;type&gt;</td>
<td>Lists the supported items for the following types:</td>
</tr>
<tr>
<td></td>
<td>• collect-configs: Predefined profile configurations that can be used</td>
</tr>
<tr>
<td></td>
<td>with collect--config option.</td>
</tr>
<tr>
<td></td>
<td>• predefined-events: List of the supported predefined events that can be</td>
</tr>
<tr>
<td></td>
<td>used with collect--event option.</td>
</tr>
<tr>
<td></td>
<td>• pmu-events: Raw PMC events that can be used with collect--event option.</td>
</tr>
<tr>
<td></td>
<td>Alternatively, info --pmu-event all can be used to print information of</td>
</tr>
<tr>
<td></td>
<td>all the supported events.</td>
</tr>
<tr>
<td></td>
<td>• view-configs: List the supported data view configurations that can be</td>
</tr>
<tr>
<td></td>
<td>used with report-view option.</td>
</tr>
<tr>
<td>--collect-config &lt;name&gt;</td>
<td>Displays the details of the given profile configuration used with collect--config &lt;name&gt; option. Use info --list collect-configs command for the details on the supported profile configurations.</td>
</tr>
<tr>
<td>--view-config &lt;name&gt;</td>
<td>Displays the details of the given view configuration used in the report</td>
</tr>
<tr>
<td></td>
<td>generation option report--view &lt;name&gt;.</td>
</tr>
<tr>
<td></td>
<td>Use info --list view-configs command for the details on the supported</td>
</tr>
<tr>
<td></td>
<td>data view configurations.</td>
</tr>
<tr>
<td>--pmu-event &lt;event&gt;</td>
<td>Displays the details of the given pmu event. Use command info --list pmu--event &lt;event&gt; for the list of supported PMC events.</td>
</tr>
<tr>
<td>--system</td>
<td>Displays the processor information of this system.</td>
</tr>
</tbody>
</table>

Following table lists the Linux specific info command options:

#### Table 27. Info Command - Linux Specific Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--list &lt;type&gt;</td>
<td>Lists the supported items for the following types:</td>
</tr>
<tr>
<td></td>
<td>• cacheline-events: List of event aliases to be used with report--sort-by</td>
</tr>
<tr>
<td></td>
<td>option for cache analysis.</td>
</tr>
<tr>
<td></td>
<td>• ostrace-events: List of OS Trace events that can be used with --trace os</td>
</tr>
<tr>
<td></td>
<td>option.</td>
</tr>
<tr>
<td></td>
<td>• gpu-events: List of GPU events can be used in gpu profile configuration.</td>
</tr>
<tr>
<td>--bpf</td>
<td>Displays details of the BPF support and BCC Installation.</td>
</tr>
</tbody>
</table>
6.6.2 Examples

Use the following commands to:

- Print the system details:
  ```
  C:\> AMDuProfCLI.exe info --system
  ```

- Print the list of predefined profiles:
  ```
  C:\> AMDuProfCLI.exe info --list collect-configs
  ```

- Print the list of PMU events:
  ```
  C:\> AMDuProfCLI.exe info --list pmu-events
  ```

- Print the list of predefined report views:
  ```
  C:\> AMDuProfCLI.exe info --list view-configs
  ```

- Print details of predefined profile such as “assess_ext”:
  ```
  C:\> AMDuProfCLI.exe info --collect-config assess_ext
  ```

- Print the details of the pmu-event such as PMCx076:
  ```
  C:\> AMDuProfCLI.exe info --pmu-event pmcx76
  ```

- Print details of view configuration such as ibs_op_overall:
  ```
  C:\> AMDuProfCLI.exe info --view-config ibs_op_overall
  ```
Chapter 7   Performance Analysis

7.1   CPU Profiling

AMD uProf profiler follows a statistical sampling-based approach to collect profile data to identify the performance bottlenecks in the application.

Profile data is collected using one of the following approaches:

• Timer Based Profiling (TBP) — to identify the hotspots in the profiled applications.

• Event Based Profiling (EBP) — sampling based on Core PMC events to identify micro-architecture related performance issues in the profiled applications.

• Instruction based Sampling (IBS) — precise instruction-based sampling.

• Call-stack Sampling

• Secondary Profile Data (Windows only)
  – Thread concurrency
  – Thread names

• Profile Scope
  – Per-Process — launch an application and profile that process and its children.
  – System-wide — profile all the running processes and/or kernel.

Attach to an existing application (Native applications only)

• Profile mode — profile data is collected when the application is running in User and/or Kernel mode.

• Profiles

• C, C++, Java, .NET, FORTRAN, and Assembly applications

• Various software components — applications, dynamically linked/loaded modules, driver, and OS kernel modules

• Profile data is attributed at various granularities
  – Process, Thread, Load Module, Function, Source line, or Disassembly
  – To correlate the profile data to Function and Source line, debug information emitted by the compiler is required
  – C++ and Java in-lined functions

• Processed profile data is stored in a database that can be used for generating a report later.

• Profile reports are available in comma-separated-value (CSV) format to use with spreadsheets.
• AMDuProfCLI, the command-line-interface can be used to configure a profile run, collect the profile data, and generate the profile report.
  – Collect option to configure and collect the profile data
  – Report option to process the profile data and to generate the profile report
• AMDuProf GUI can be used to:
  – Configure a profile run
  – Start the profile run to collect the performance data
  – Analyze the performance data to identify potential bottlenecks

AMDuProf GUI has various UI elements to analyze and view the profile data at various granularities:
  – Hot spots summary
  – Thread concurrency graph (Windows only and requires admin privileges)
  – Process and function analysis
  – Source and disassembly analysis
  – Flame Graph — a stack visualizer based on collected call-stack samples
  – Call Graph — butterfly view of callgraph based on call-stack samples
  – HPC — to analyze OpenMP profile data
  – Cache Analysis — to analyze the hot cache lines that are false shared
• Profile Control API to selectively enable and disable profiling from the target application by instrumenting it, to limit the scope of the profiling.

7.2 Analysis with Time-based Profiling

In this analysis, the profile data is periodically collected based on the specified OS timer interval. It is used to identify the hotspots of the profiled applications that are consuming the most time. These hotspots are good candidates for further investigation and optimization.

7.2.1 Configuring and Starting Profile

Complete the following steps to configure and start a profile:

1. Click PROFILE > Start Profiling to navigate to the Select Profile Target screen.
2. Select the required profile target, click the Next button.
   The Select Profile Type screen is displayed.
3. From the **Select Profile Type** drop-down, select the **CPU Profile** from the drop-down.

   The following screen is displayed:

   ![Figure 25. Time-based Profile – Configure](image)

4. Select **Time-based Sampling** in the left vertical pane.

5. Click **Advanced Options** to enable call-stack, set symbol paths (if the debug files are in different locations) and other options. Refer the section “Source and Assembly” on page 58 section for more information on this screen.

6. Once all the options are set, the **Start Profile** button at the bottom will be enabled and you can click on it to start the profile.

   After the profile initialization the profile data collection screen is displayed.

### 7.2.2 Analyzing Profile Data

Complete the following steps to analyze the profile data:

1. When the profiling stops, the collected raw profile data will be processed automatically and you the **Hot Spots** screen of the **Summary** page is displayed. The hotspots are shown for the **Timer** samples. Refer the section “Overview of Performance Hotspots” on page 53 for more information on this screen.

2. Click **ANALYZE** button on the top horizontal navigation bar to go to the **Function HotSpots** screen. Refer the section “Function HotSpots” on page 55 for more information on this screen.
3. Click **ANALYZE > Metrics** to display the profile data table at various granularities - Process, Load Modules, Threads, and Functions. Refer the section “Process and Functions” on page 56 for more information on this screen.

4. Double-click any entry on the **Functions** table in **Metrics** screen to load the source tab for that function in **SOURCES** page. Refer the section “Source and Assembly” on page 58 for more information on this screen.

### 7.3 Analysis with Event-based Profiling

In this profile, AMD uProf uses the PMCs to monitor the various micro-architectural events supported by the AMD x86-based processor. It helps to identify the CPU and memory related performance issues in profiled applications.

#### 7.3.1 Configuring and Starting Profile

Complete the following steps to configure and start a profile:

1. Click **PROFILE > Start Profiling** to navigate to the **Select Profile Target** screen.

2. Select the required profile target, click the **Next** button.

The **Start Profiling** screen is displayed as follows:

![Event-based Profile – Configure](image)

3. From the **Select Profile Type** drop-down, select the CPU Profile.
4. Select **Assess Performance** in the left vertical pane. Refer the section “Predefined Sampling Configuration” on page 25 for EBP based predefined sampling configurations.

5. Click **Advanced Options** to enable call-stack, set symbol paths (if the debug files are in different locations) and other options. Refer the section “Advanced Options” on page 50 for more information on this screen.

6. Once all the options are set, the **Start Profile** button at the bottom will be enabled. Click it to start the profile.
   
   After the profile initialization the profile data collection screen is displayed.

### 7.3.2 Analyzing Profile Data

Complete the following steps to analyze the profile data:

1. When the profiling stops, the collected raw profile data will be processed automatically and you the **Hot Spots** screen of the **Summary** page is displayed. Refer the section “Overview of Performance Hotspots” on page 53 for more information on this screen.

2. Click the **ANALYZE** button on the top horizontal navigation bar to go to the **Function HotSpots** screen. Refer the section “Function HotSpots” on page 55 for more information on this screen.

3. Click **ANALYZE > Metrics** to display the profile data table at various granularities - Process, Load Modules, Threads, and Functions. Refer the section “Process and Functions” on page 56 for more information on this screen.

4. Double-click any entry on the **Functions** table in **Metrics** screen to load the source tab for that function in **SOURCES** page. Refer the section “Source and Assembly” on page 58 for more information on this screen.

### 7.4 Analysis with Instruction-based Sampling

In this profile, AMD uProf uses the IBS supported by the AMD x64-based processor to diagnose the performance issues in hot spots. It collects data on how instructions behave on the processor and in the memory sub-system.

#### 7.4.1 Configuring and Starting Profile

Complete the following steps to configure and start a profile:

1. Click **PROFILE > Start Profiling** to navigate to the **Select Profile Target** screen.

2. Select the required profile target, click the **Next** button.

3. From the **Select Profile Type** drop-down, select the **CPU Profile**.

4. Select **Instruction-based Sampling** in the left vertical pane. Refer the section “Predefined Sampling Configuration” on page 25 for IBS based predefined sampling configurations.
Chapter 7 Performance Analysis

5. Click Advanced Options to enable call-stack, set symbol paths (if the debug files are in different locations) and other options. Refer the section “Advanced Options” on page 50 for more information on this screen.

6. Once all the options are set, the Start Profile button at the bottom will be enabled. Click it to start the profile.

After the profile initialization the profile data collection screen is displayed.

7.4.2 Analyzing Profile Data

Complete the following steps to analyze the profile data:

1. When the profiling stops, the collected raw profile data will be processed automatically and you the Hot Spots screen of the Summary page is displayed. Refer the section “Overview of Performance Hotspots” on page 53 for more information on this screen.

2. Click the ANALYZE button on the top horizontal navigation bar to go to the Function HotSpots screen. Refer the section “Process and Functions” on page 56 for more information on this screen.

3. Click ANALYZE > Metrics to display the profile data table at various granularities - Process, Load Modules, Threads, and Functions. Refer the section “Process and Functions” on page 56 for more information on this screen.

4. Double-click any entry on the Functions table in Metrics screen to load the source tab for that function in SOURCES page. Refer the section “Source and Assembly” on page 58 for more information on this screen.

7.5 Analysis with Call Stack Samples

The call stack samples can be collected for C, C++, and Java applications with all the CPU profile types. These samples will be used to provide Flame Graph and Call Graph window.

To enable call stack sampling, complete the following steps:

1. Select profile target and profile type.
2. Click on **Advanced Options** button to turn on the **Enable CSS** option in **Call Stack Options** pane as follows:

![Figure 27. Start Profiling - Advanced Options](image)

Refer the section “Advanced Options” on page 50 for more information on this screen.

**Note:** If the application is compiled with higher optimization levels and frame pointers are not displayed, **Enable FPO** option can be turned on. On Linux, this will increase the size of the raw profile file size.

### 7.5.1 Flame Graph

Flame Graph provides a stack visualizer based on call stack samples. The **Flame Graph** is available in the **ANALYZE** page to analyze the call stack samples to identify hot call-paths. To access it, navigate to **ANALYZE > Flame Graph** in the left vertical pane.

Refer the section “Flame Graph” on page 59 for more information on this screen.
The following figure shows a sample flame graph:

![Flame Graph Example]

**Figure 28. ANALYZE - Flame Graph**

The flame graph can be displayed based on the Process IDs and Counters drop-downs. Also, it has the function search box to search and highlight the given function name.

### 7.5.2 Call Graph

Call Graph provides a butterfly view of call graph based on call-stack samples. The Call Graph screen will be available in ANALYZE page to analyze the call-stack samples to identify hot call-paths. To access it, click ANALYZE > Call Graph in the left vertical pane.

Refer the section “Call Graph” on page 61 for more information on this screen.
The following figure shows a sample call graph:

![Call Graph](image)

**Figure 29. ANALYZE - Call Graph**

You can browse the data based on **Process IDs** and **Counters** drop-downs. The top central table displays call-stack samples for each function. Click on any function to update the bottom two **Caller(s)** and **Callee(s)** tables. These tables display the callers and callees respectively of the selected function.

### 7.6 Profiling a Java Application

AMD uProf supports Java application profiling running on JVM. To support this, it uses JVM Tool Interface (JVMTI).

AMD uProf provides JVMTI Agent libraries: `AMDJvmtiAgent.dll` on Windows and `libAMDJvmtiAgent.so` on Linux. This JvmtiAgent library must be loaded during start up of the target JVM process.

#### 7.6.1 Launching a Java Application

If the Java application is launched by AMD uProf, the tool would pass the AMDJvmtiAgent library to JVM using Java `-agentpath` option. AMD uProf would be able to collect the profile data and attribute the samples to interpreted Java functions.

To profile a Java application, use the following sample command:

```
$ ./AMDuProfCLI collect --config tbp -w <java-app-dir> <path-to-java.exe> <java-app-main>
```

To generate report, pass the following source file path:

```
$ ./AMDuProfCLI report --src-path <path-to-java-app-source-dir> -i <raw-data-file-path>
```
7.6.2  Attaching a Java Process to Profile

AMD uProf cannot attach JvmtiAgent dynamically to an already running JVM. Hence, for any JVM process profiled by attach-process mechanism, AMD uProf cannot capture any class information, unless the JvmtiAgent library is loaded during JVM process start up.

To profile an already running Java process, pass -agentpath <path to agent lib> option while launching Java application. So that, AMD uProf can attach to the Java PID to collect profile data later on.

For a 64-bit JVM on Linux:

$ java -agentpath:<AMDuProf-install-dir/bin/ProfileAgents/x64/libAMDJvmtiAgent.so> <java-app-launch-options>

For a 64-bit JVM on Windows:

C:\> java -agentpath:C:\ProgramFiles\AMD\AMDuProf\bin\ProfileAgents\x64\AMDJvmtiAgent.dll> <java-app-launch-options>

Keep a note of the process id (PID) of the above JVM instance. Then, launch AMD uProf GUI or AMD uProf CLI to attach to this process and profile.

7.6.3  Java Source View

AMD uProf will attribute the profile samples to Java methods and the source tab will show and the Java source lines with the corresponding samples attributed to them.

Refer the section “Source and Assembly” on page 58 for more information on source screen.
The following figure shows the source view of the Java method:

![Java Method - Source View](image)

**Figure 30. Java Method - Source View**

### 7.6.4 Java Call Stack Profile

To collect call stack for profiling Java application, use the following command:

```
$ ./AMDuProfCLI collect --config tbp -g -w <java-app-dir> <path-to-java.exe> <java-app-main>
```
The following figure shows the flame graph of a Java application:

![Flame Graph of a Java Application](image)

**Figure 31. Java Application - Flame Graph**

### 7.7 Cache Analysis

The **Cache Analysis** uses IBS OP samples to detect the hot false sharing cache lines in multi-threaded and multi-process with shared memory applications.

At a high-level, this will feature will report:

- The cache lines where there is a potential false sharing
- Offsets where those accesses occur, readers and writers to those offsets
- PID, TID, Function Name, Source File, and Line Number for those reader and writers
- Load latency for the loads to those cache lines
7.7.1 Supported Metrics

The following IBS OP derived metrics are used to generate false cache sharing report:

Table 28. IBS OP Derived Metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS_LOAD_STORE</td>
<td>Total Loads and stores sampled</td>
</tr>
<tr>
<td>IBS_LOAD</td>
<td>Total Loads</td>
</tr>
<tr>
<td>IBS_STORE</td>
<td>Total Stores</td>
</tr>
<tr>
<td>IBS_DC_MISS_LATENCY</td>
<td>Accumulated load latencies for the loads to cache lines</td>
</tr>
<tr>
<td>IBS_LOAD_DC_L2_HIT</td>
<td>Load operations hit in data cache or L2 cache</td>
</tr>
<tr>
<td>IBS_NB_LOCAL_CACHE_MODIFIED</td>
<td>Loads that were serviced from the local cache (L3) and the cache hit state was Modified</td>
</tr>
<tr>
<td>IBS_NB_LOCAL_CACHE_OWNED</td>
<td>Loads that were serviced from the local cache (L3) and the cache hit state was Owned</td>
</tr>
<tr>
<td>IBS_NB_LOCAL_CACHE_MISS</td>
<td>Loads that were missed in local cache (L3) and serviced by remote cache, local, or remote DRAM</td>
</tr>
<tr>
<td>IBS_NB_REMOTE_CACHE_MODIFIED</td>
<td>Loads that were serviced from the remote cache (L3) and the cache hit state was Modified</td>
</tr>
<tr>
<td>IBS_NB_REMOTE_CACHE_OWNED</td>
<td>Loads that were serviced from the remote cache (L3) and the cache hit state was Owned</td>
</tr>
<tr>
<td>IBS_NB_LOCAL_DRAM</td>
<td>Loads that hit in local memory (Memory channels attached to local socket or local CCD)</td>
</tr>
<tr>
<td>IBS_NB_REMOTE_DRAM</td>
<td>Loads that hit in remote memory (Memory channels attached to remote socket or other CCDs in the local socket)</td>
</tr>
<tr>
<td>IBS_STORE_DC_MISS</td>
<td>Store operations missed in data cache</td>
</tr>
</tbody>
</table>

7.7.2 Cache Analysis Using GUI

Configuring and Starting Profile

To perform cache analysis, complete the following steps:

1. Selecting profile target.
2. Select Cache Analysis profile type in Select Profile Type page.
3. Start the profile.
Analyzing the Report

After the profile completion, navigate to Cache Analysis page in MEMORY tab to analyze the profile data. This page shows the cache-lines and it offsets with the associated metric values.

![Cache Analysis](image)

**Figure 32. Cache Analysis (from AMD uProf 3.4)**

The Cache Analysis screen has the following options:

- **Group By** drop-down decides how the cache-line samples are grouped in the detailed table. It has the option Cache Line Offset.

- **Show Values By** drop-down allows you to show the value in sample count.

### 7.7.3 Cache Analysis Using CLI

The CLI has a config type called “memory” to cache the analysis data. Run the following command to collect the profile data:

```
$ AMDuProfCLI collect --config memory -o /tmp/cache_analysis <target app>
```

This command will launch the program and collect the profile data required to generate the cache analysis report. The raw profile data file is created in `/tmp/cache_analysis/AMDuProf-IBS_<timestamp>/` directory.

**Report Generation and Analysis**

Use the following CLI command to generate the cache analysis report:

```
$ AMDuProfCLI report -i /tmp/cache_analysis/AMDuProf-IBS_<timestamp>/
```

This will generate a CSV report in `/tmp/cache_analysis/AMDuProf-IBS_<timestamp>/report.csv` and it will have the following sections:

- **SHARED DATA CACHELINE SUMMARY**: Lists the summary values of all the metrics.
- **SHARED DATA CACHELINE REPORT**: Lists the cache lines and the associated summary values of the metrics.
• **SHARED DATA CACHELINE DETAIL REPORT:** Lists the following:
  - The cache lines having a potential false sharing
  - Offsets where those accesses occur, readers and writers to those offsets
  - PID, TID, Function Name, Source File, and Line Number for those reader and writers
  - Load latency for the loads to those cache lines
  - Supported metrics

By default, the generated report will have a cut-off limit of 10 entries for each of the above-mentioned sections. To include more entries, use the option `--limit-cacheinfo <cutoff-value>` with report command:

```
$ AMDuProfCLI report --limit-cacheinfo <cutoff-value> -i /tmp/cache_analysis/AMDuProf-IBS_<timestamp>/
```

Following figure shows the **Cache Analysis** summary sections:

![Cache Analysis - Summary Sections](image)

**Figure 33.** Cache Analysis - Summary Sections (from AMD uProf 3.4)
Following figure shows the **Cache Analysis** detailed report:

![Cache Analysis - Detailed Report](image)

**Figure 34. Cache Analysis - Detailed Report (from AMD uProf 3.4)**

By default, the metrics are sorted by IBS DC_MISS_LATENCY metric. Use any of the following metric with the `--sort-by` option to change the sorting by order:

**Table 29. Sort-by Metric**

<table>
<thead>
<tr>
<th>Sort-by Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldst-count</td>
<td>Total Loads and stores sampled</td>
</tr>
<tr>
<td>ld-count</td>
<td>Total Loads</td>
</tr>
<tr>
<td>st-count</td>
<td>Total Stores</td>
</tr>
<tr>
<td>cache-hitm</td>
<td>Loads that were serviced either from the local or remote cache (L3) and the cache hit state was Modified.</td>
</tr>
<tr>
<td>lcl-cache-hitm</td>
<td>Loads that were serviced from the local cache (L3) and the cache hit state was Modified</td>
</tr>
<tr>
<td>rmt-cache-hitm</td>
<td>Loads that were serviced from the remote cache (L3) and the cache hit state was Modified.</td>
</tr>
<tr>
<td>lcl-dram-hit</td>
<td>Loads that hit in local memory (Memory channels attached to local socket or local CCD)</td>
</tr>
<tr>
<td>rmt-dram-hit</td>
<td>Loads that hit in remote memory (Memory channels attached to remote socket or other CCDs in the local socket)</td>
</tr>
<tr>
<td>l3-miss</td>
<td>Loads that are missed in local cache (L3) and serviced by remote cache, local or remote DRAM.</td>
</tr>
<tr>
<td>st-dc-miss</td>
<td>Store operations missed in data cache</td>
</tr>
</tbody>
</table>
7.8 Custom Profile

Apart from the predefine configurations, you can choose the required events to profile. To perform the custom profile, complete the following steps:

7.8.1 Configuring and Starting Profile

1. Click PROFILE > Start Profiling to navigate to the Select Profile Target screen.
2. Selecting the required profile target and click the Next button.
   The Select Profile Type screen is displayed.
3. From the Select Profile Type drop-down, select CPU Profile as follows:

![Image of Select Profile Type screen]

Figure 35. Start Profiling - Custom Profile

4. Select Custom Profile in the left vertical pane.
5. Click Advanced Options to enable call-stack, set symbol paths (if the debug files are in different locations) and other options. Refer the section “Advanced Options” on page 50 for more information on this screen.
6. Once all the options are set, the Start Profile button at the bottom will be enabled. Click it to start the profile.
   After the profile initialization the profile data collection screen is displayed.
7.8.2 Analyzing Profile Data

Complete the following steps to analyze the profile data:

1. When the profiling stops, the collected raw profile data will be processed automatically and you the Hot Spots screen of the Summary page is displayed. Refer the section “Overview of Performance Hotspots” on page 53 for more information on this screen.

2. Click the ANALYZE button on the top horizontal navigation bar to go to the Function HotSpots screen. Refer the section “Function HotSpots” on page 55 for more information on this screen.

3. Click ANALYZE > Metrics to display the profile data table at various granularities - Process, Load Modules, Threads, and Functions. Refer the section “Process and Functions” on page 56 for more information on this screen.

4. Double-click any entry on the Functions table in Metrics screen to load the source tab for that function in SOURCES page. Refer the section “Source and Assembly” on page 58 for more information on this screen.

7.9 Advisory

7.9.1 Confidence Threshold

The metric with low number of samples collected for a program unit either due to multiplexing or statical sampling will be grayed out. A few points to remember are:

- This is applicable to SW Timer and Core PMC based metrics.
- This confidence threshold value can be set through Preferences section in SETTINGS page.
7.9.2 Issue Threshold

Highlight the CPI metric cells exceeding the specific threshold value (>1.0). Those cells will be highlighted in pink to show them as potential performance problem as follows:

Figure 36. CPI Metric - Threshold-based Performance Issue

7.10 ASCII Dump of IBS Samples

For some scenarios, it would be useful to analyze the ASCII dump of IBS OP profile samples. To do so, complete the following steps:

1. To collect the IBS OP samples, run:

   ```
   C:\> AMDuProfCLI.exe collect -e event=ibs-op,interval=100000,loadstore,ibsop-count-control=1 -a --data-buffer-count 20480 -d 250 -o C:\temp\ 
   ```

2. Once the raw file is generated, run the following command to translate and get the ASCII dump of IBS OP samples:

   ```
   C:\> AMDuProfCLI.exe translate --ascii event-dump -i C:\temp\AMDuProf-IBS_<timestamp>\C:\temp\AMDuProf-IBS_<timestamp>\IbsOpDump.csv
   ```

The text file that containing ASCII dump of the IBS OP samples is generated:

```
C:\temp\AMDuProf-IBS_<timestamp>\IbsOpDump.csv
```
3. During collection the following control knobs are available:
   
   `-e event=ibs-op,interval=100000,loadstore,ibsop-count-control=1`
   
   Where:
   
   - `interval` denotes sampling interval
   - `loadstore` denotes collect only the load & store ops (Windows only option)
   - `ibsop-count-control=1` represents count dispatched micro-ops (0 for “count clock cycles”)
   - `--data-buffer-count 1024` represents the number of per-core data buffers to allocate

   In case, there are too many missing records, try the following:
   
   - Increase the sampling interval
   - Increase the data buffer count
   - Reduce the number of cores profiled

7.11 Limitations

CPU profiling in AMD uProf 3.5 has the following limitations:

- CPU profiling expects the profiled application executable binaries must not be compressed or obfuscated by any software protector tools, for example, VMProtect.

- In case of AMD EPYC™ 1st generation B1 parts, only one PMC register is used at a time for Core PMC event-based profiling (EBP).
Chapter 8  Performance Analysis (Linux)

This chapter explains the Linux specific performance analysis models.

8.1  OpenMP Analysis

The OpenMP API uses the fork-join model of parallel execution. The program starts with a single master thread to run the serial code. When a parallel region is encountered, multiple threads perform the implicit or explicit tasks defined by the OpenMP directives. At the end of that parallel region, the threads join at the barrier and only the master thread continues to execute.

When the threads execute the parallel region code, they should utilize all the available CPU cores and the CPU utilization should be maximized. But, the threads wait without doing anything useful due to several reasons:

- **Idle**: A thread finishes its task within the parallel region and waits at the barrier for the other threads to complete.
- **Sync**: If locks are used inside the parallel region, threads can wait on synchronization locks to acquire the shared resource.
- **Overhead**: The thread management overhead.

The OpenMP analysis helps to trace the activities performed by OpenMP threads, their states, and provides the thread state timeline for parallel regions to analyze the performance issues.

Support Matrix

The following table shows the support matrix:

<table>
<thead>
<tr>
<th>Component</th>
<th>Supported Versions</th>
<th>Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMP Spec</td>
<td>OpenMP v5.0</td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>LLVM 8, 9, 10, 11</td>
<td>C and C++</td>
</tr>
<tr>
<td></td>
<td>AOCC 2.1, 2.2, 2.3, 3.0</td>
<td>C, C++, and Fortran</td>
</tr>
<tr>
<td></td>
<td>ICC 19.1, 2021.1.1</td>
<td>C, C++, and Fortran</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 18.04 LTS, 20.04 LTS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RHEL 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CentOS 8</td>
<td></td>
</tr>
</tbody>
</table>
Prerequisite

Compile the OpenMP application using a supported compiler (on a supported platform) with the required compiler options to enable OpenMP.

8.1.1 Profiling OpenMP Application using GUI

Configuring and Starting a Profile

Complete the following steps to enable the OpenMP profiling:

1. Select the profile target and profile type.
2. Click the Advanced Options button.
3. In Enable OpenMP Tracing pane, turn on the Enable OpenMP Tracing option in, as shown in the following image:

![Figure 37. Enable OpenMP Tracing](image)

Analyzing the OpenMP Report

After the profile completion, navigate to the HPC page to analyze the OpenMP tracing data. You can use the left side vertical pane on this page to navigate through the following views:

- **Overview** shows the quick details about the runtime. The following image shows the Overview page:

![Figure 38. HPC - Overview (from AMD uProf 3.4)](image)
• **Parallel Regions** shows the summary of all the parallel regions. This tab is useful to quickly understand which parallel region might be load imbalanced. Double-click on the region names to open the *Regions Detailed Analysis* page.

Figure 39.  **HPC - Parallel Regions (from AMD uProf 3.4)**

### 8.1.2 Profiling OpenMP Application Using CLI

#### Collect Profile Data

Use the following command to profile an OpenMP application using AMD uProf CLI:

```
$ ./AMDuProfCLI collect --trace openmp --config tbp -o /tmp/myapp_perf <openmp-app>
```

While performing the regular profiling, add option `--trace openmp` or `--omp` to enable OpenMP profiling. This command will launch the program and collect the profile data required to generate the OpenMP analysis report.

#### Generate Profile Report

You can generate a CSV report using the `AMDuProfCLI report` command. Any additional option is not required for the OpenMP report generation. AMD uProf checks for the availability of any OpenMP profiling data and includes it in the report if available.

The following command will generate a CSV report in `/tmp/myapp_perf/<SESSION-DIR>/report.csv`:

```
$ ./AMDuProfCLI report -i /tmp/myapp_perf/<SESSION-DIR>
```
An example of the OpenMP report section in the CSV file is as follows:

```
<table>
<thead>
<tr>
<th>Region</th>
<th>Imbalance Time</th>
<th>Imbalance Time (%)</th>
<th>Threads</th>
<th>Sync Time</th>
<th>Overhead</th>
<th>Work Time</th>
<th>Total Time</th>
<th>Overlap Schedule Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>outter_sequence_coroutines SMP</td>
<td>0.000000</td>
<td>0.000000</td>
<td>3</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>outter_sequence_coroutines SMP</td>
<td>0.000000</td>
<td>0.000000</td>
<td>3</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>outter_sequence_coroutines SMP</td>
<td>0.000000</td>
<td>0.000000</td>
<td>3</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>outter_sequence_coroutines SMP</td>
<td>0.000000</td>
<td>0.000000</td>
<td>3</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
</tbody>
</table>
```

**Figure 40. An OpenMP Report**

It has following sub-sections:

- **OpenMP Overview**

- **OpenMP PARALLEL-REGION METRIC** helps in understanding the imbalanced region, that is, a region with less total work time with respect to its total time.

- **OpenMP THREAD METRIC** helps in understanding how each thread spent its time in the parallel region. If a thread spends too much time on non-work activity, the parallel region should be optimized further to improve the work time of each thread in that region.

### 8.1.3 Environment Variables

- **AMDUPROF_MAX_PR_INSTANCES** – Set the max number of unique parallel regions to be traced. The default value is 512.

- **AMDUPROF_MAX_PR_INSTANCE_COUNT** – Set the max number of times one unique parallel region to be traced. The default value is 512.

### 8.1.4 Limitations

The following features not supported in this release:

- OpenMP profiling with system-wide profiling scope.
• Loop chunk size and schedule type when the parameters are specified using schedule clause. In such as case, it shows the default values (1 and Static).

• Nested parallel regions.

• GPU offloading and related constructs.

• Callstack for individual OpenMP threads.

• OpenMP profiling on Windows and FreeBSD platforms.

• Applications with static linkage of OpenMP libraries.

• Attaching to running OpenMP application.

8.2 MPI Profiling

The MPI programs launched through `mpirun` or `mpiexec` launcher programs can be profiled by AMD uProf. To profile the MPI applications and analyze the data, complete the following the steps:

1. Collect the profile data using CLI collect command.
2. Process the profile data using CLI translate command which will generate the profile database.
3. Import the profile database in the GUI or generate the CSV report using CLI report command.

Support Matrix

The MPI profiling supports the following components and the corresponding versions:

<table>
<thead>
<tr>
<th>Component</th>
<th>Supported Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI Spec</td>
<td>MPI v3.1</td>
</tr>
<tr>
<td>MPI Libraries</td>
<td>Open MPI v4.1.0</td>
</tr>
<tr>
<td></td>
<td>MPICH v3.4.1</td>
</tr>
<tr>
<td></td>
<td>ParaStation MPI v5.4.8</td>
</tr>
<tr>
<td></td>
<td>Intel® MPI 2021.1</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 18.04 LTS, 20.04 LTS</td>
</tr>
<tr>
<td></td>
<td>RHEL 8</td>
</tr>
<tr>
<td></td>
<td>CentOS 8</td>
</tr>
</tbody>
</table>

8.2.1 Collecting Data Using CLI

The MPI jobs are launched using MPI launchers such as `mpirun` and `mpiexec`. You must use AMDuProfCLI to collect the profile data for an MPI application.

The MPI job launch through `mpirun` uses the following syntax:

```bash
$ mpirun [options] <program> [<args>]
```
AMDuProfCLI is launched using \texttt{<program>} and the application is launched using the AMDuProfCLI's arguments. So, use the following syntax to profile an MPI application using AMDuProfCLI:

\begin{verbatim}
$ mpirun [options] AMDuProfCLI [options] <program> [<args>]
\end{verbatim}

The MPI profiling specific AMDuProfCLI options:

- The \texttt{--mpi} option is to profile MPI application. The AMDuProfCLI will collect some additional meta data from MPI processes.
- \texttt{--output-dir <output dir>} specifies the path to a directory in which the profile files are saved. A session directory will be created within the \texttt{<output dir>} containing all the data collected from all the ranks.

A typical command uses the following syntax:

\begin{verbatim}
$ mpirun -np <n> /tmp/AMDuProf/bin/AMDuProfCLI collect
--config <config-type> --mpi --output-dir <output_dir> [mpi_app] [mpi_app_options]
\end{verbatim}

If an MPI application is launched on multiple nodes, AMDuProfCLI will profile all the MPI rank processes running on all the nodes. You can either analyze the data for processes ran on one/many/all node(s).

**Method 1 - Profile All the Ranks On Single/Multiple Node(s)**

To collect profile data for all the ranks running on a single node, execute the following commands:

\begin{verbatim}
$ mpirun -np 16 /tmp/AMDuProf/bin/AMDuProfCLI collect --config tbp
--mpi --output-dir /tmp/myapp-perf myapp.exe
\end{verbatim}

To collect profile data for all the ranks in multiple nodes, use the options \texttt{-H / --host mpirun} or specify \texttt{-hostfile <hostfile>}: 

\begin{verbatim}
$ mpirun -np 16 -H host1,host2 /tmp/AMDuProf/bin/AMDuProfCLI collect
--config tbp --mpi --output-dir /tmp/myapp-perf myapp.exe
\end{verbatim}

**Method 2 - Profiling Specific Rank(s)**

To profile only a specify rank running on a host2, execute the following commands:

\begin{verbatim}
$ export AMDUPROFCLI_CMD=/tmp/AMDuProf/bin/AMDuProfCLI collect --config tbp --mpi --output-dir /tmp/myapp-perf
$ mpirun -np 4 -host host1 myapp.exe : -host host2 -np 2
$AMDUPROFCLI_CMD myapp.exe
\end{verbatim}

To profile only a single rank in setup where 256 ranks running on 2 hosts (128 ranks per host):

\begin{verbatim}
$ mpirun -host host1:128 -np 1 $AMDUPROFCLI_CMD myapp.exe : -host host2:128,host1:128 -np 255
--map-by core myapp.exe
\end{verbatim}
Method 3 – Using MPI Config File

The `mpirun` also takes config file as an input and the AMDuProfCLI can be used with the config file to profile the MPI application.

Config file (`myapp_config`):

```bash
#MPI - myapp config file
-host host1 -n 4 myapp.exe
-host host2 -n 2 /tmp/AMDuProf/bin/AMDuProfCLI collect --config tbp --mpi \
-host-dir /tmp/myapp-perf myapp.exe
```

To run this config to collect data only for the MPI processes running on host2, execute the following command:

```bash
$ mpirun --app myapp_config
```

### 8.2.2 Analyzing the Data with CLI

The data collected for MPI processes can be analyzed using the CSV reported by the AMDuProfCLI’s report command. The generated reported is saved to the file `report.csv` in the `<output-dir>/<SESSION-DIR>` folder.

Following are the reporting options for the CLI:

- **Generate a report for all the MPI processes ran on the localhost (for example, host1) in which the MPI launcher was launched (using the new option `--input-dir`):**

  ```bash
  $ AMDuProfCLI report --input-dir /tmp/myapp-perf/<SESSION-DIR> --host host1
  ```

  Option `--host` is not mandatory to create the report file for the localhost.

- **Generate a report for all the MPI processes ran on another host (for example, host2) in which the MPI launcher was not launched:**

  ```bash
  $ AMDuProfCLI report --input-dir /tmp/myapp-perf/<SESSION-DIR> --host host2
  ```

- **Generate a report for all the MPI processes ran on all the hosts:**

  ```bash
  $ AMDuProfCLI report --input-dir /tmp/myapp-perf/<SESSION-DIR> --host all
  ```

### 8.2.3 Analyze the Data with GUI

To analyze the profile data in the GUI, complete the following steps:

1. To generate the profile database, refer “Analyzing the Data with CLI” on page 114.
2. To import the profile database, refer “Importing Profile Database” on page 62.

### 8.2.4 Limitations

The MPI environment parameters such as **Total number of ranks** and **Number of ranks running on each node** are currently supported only for OpenMPI.
### 8.3 Profiling Linux System Modules

To attribute the samples to the system modules (for example, glibc and libm), AMD uProf uses the corresponding debug info files. The Linux distros do not contain the debug info files, but most of the popular distros provide options to download the debug info files.

Refer the following resources for more information on how to download the debug info files:

- SLES/OpenSUSE ([https://www.suse.com/support/kb/doc/?id=3074997](https://www.suse.com/support/kb/doc/?id=3074997))

Ensure that you download the debug info files for the required system modules for the required Linux distros before starting the profiling.

### 8.4 Profiling Linux Kernel

To profile and analyze the Linux kernel modules and functions, do the following:

1. Enable the kernel symbol resolution.
2. Do one of the following:
   - Download and install kernel debug symbol packages and source.
   - Build Linux kernel with debug symbols.

After the kernel debug info is available in the default path, AMD uProf automatically locates and utilizes that debug info to show the kernel sources lines and assembly in the source view.

**Supported OS:** Ubuntu 18.04 LTS, Ubuntu 20.04 LTS, RHEL 7, and RHEL 8

#### 8.4.1 Enabling Kernel Symbol Resolution

To attribute the kernel samples to appropriate kernel functions, AMD uProf extracts required information from the `/proc/kallsyms` file. Exposing the kernel symbol addresses through `/proc/kallsyms` requires setting of the appropriate value to the `/proc/sys/kernel/kptr_restrict` file as follows:

- Set `/proc/sys/kernel/perf_event_paranoid` config to `-1`.
- Set `/proc/sys/kernel/kptr_restrict` to an appropriate value as follows:
  - `0`: The kernel addresses are available without any limitations.
  - `1`: The kernel addresses are available if the current user has a CAP_SYSLOG capability.
  - `2`: The kernel addresses are hidden.
Set the `perf_event_paranoid` value using one of the following:

```bash
$ sudo echo -1 > /proc/sys/kernel/perf_event_paranoid
```

or

```bash
$ sudo sysctl -w kernel.perf_event_paranoid=-1
```

Set the `kptr_restrict` value using one of the following:

```bash
$ sudo echo 0 > /proc/sys/kernel/kptr_restrict
```

or

```bash
$ sudo sysctl -w kernel.kptr_restrict=0
```

### 8.4.2 Downloading and Installing Kernel Debug Symbol Packages

On a Linux system, the `/boot` directory either contains the compressed vmlinux or uncompressed vmlinux image. These kernel files are stripped, have no symbol and debug information. If there is no debug information, AMD uProf will not be able to attribute samples to kernel functions and hence, by default, AMD uProf cannot report kernel functions.

Some Linux distros provide debug symbol files for their kernel which can be used for profiling purposes.

**Ubuntu**

Complete the following steps to download kernel debug info and source code on Ubuntu systems (verified on Ubuntu 18.04.03 LTS):

1. To trust the debug symbol signing key, execute the following commands:

   ```bash
   // Ubuntu 18.04 LTS and later:
   $ sudo apt install ubuntu-dbgsym-keyring
   // For earlier releases of Ubuntu:
   $ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
   F2EDC64DC5AEE1F6B9C621F0C88CAB6595FDFF622
   
   $ echo "deb http://ddeb.ubuntu.com $(lsb_release -cs) main restricted universe multiverse
deb http://ddeb.ubuntu.com $(lsb_release -cs)-security main restricted universe multiverse
deb http://ddeb.ubuntu.com $(lsb_release -cs)-updates main restricted universe multiverse
deb http://ddeb.ubuntu.com $(lsb_release -cs)-proposed main restricted universe multiverse"
   | \
   sudo tee -a /etc/apt/sources.list.d/ddeb.list
   
   2. Add the debug symbol repository as follows:

   ```bash
   $ echo "deb http://ddeb.ubuntu.com $(lsb_release -cs) main restricted universe multiverse
deb http://ddeb.ubuntu.com $(lsb_release -cs)-security main restricted universe multiverse
deb http://ddeb.ubuntu.com $(lsb_release -cs)-updates main restricted universe multiverse
deb http://ddeb.ubuntu.com $(lsb_release -cs)-proposed main restricted universe multiverse"
   | \
   sudo tee -a /etc/apt/sources.list.d/ddeb.list
   ```

3. Retrieve the list of available debug symbol packages:

   ```bash
   $ sudo apt update
   ```

4. Install the debug symbols for the current kernel version:

   ```bash
   $ sudo apt install --yes linux-image-$uname_r dbgsym
   ```
5. Download the kernel source

```bash
$ sudo apt source linux-image-unsigned-$\{uname -r\}
```

or

```bash
$ sudo apt source linux-image-$\{uname -r\}
```

After the kernel debug info file is downloaded, it can be found at the default path:

`$ /usr/lib/debug/boot/vmlinux-`uname -r``

**RHEL**

Follow the steps in Red Hat knowledgebase ([https://access.redhat.com/solutions/9907](https://access.redhat.com/solutions/9907)) to download the RHEL kernel debug info.

After the kernel debug info file is downloaded, it can be found at the default path:

`$ /usr/lib/debug/lib/modules/`uname -r`/vmlinux`

### 8.4.3 Build Linux kernel with Debug Symbols

If the debug symbol packages are not available for pre-built kernel images, then analyzing the kernel functions at the source level requires a recompilation of the Linux kernel with debug flag enabled.

### 8.4.4 Analyzing Hotspots in Kernel Functions

If the debug info for the kernel modules is available, any subsequent CPU performance analysis will attribute the kernel space samples appropriately to `vmlinux` module and display the hot kernel functions. Otherwise, kernel samples will be attributed to `kernel.kallsyms_text` module.

During the hotspot analysis, do consider the following:

- If you see `vmlinux` module, then you should be able to analyze the performance data for kernel functions in the Source view and IMIX view in the GUI. The CLI should also be able to generate source level report and IMIX report for the kernel.
- If the source is downloaded and copied to the expected path, then you should be able to see the kernel source lines in GUI and CLI.
- Passing of kernel debug file path and passing of kernel source path is not recommended as that might lead to performance issues.

### 8.4.5 Linux Kernel Callstack Sampling

In System-wide profile, the callstack samples can be collected for kernel functions. For example, the following command will collect the kernel callstack:

```bash
# AMDuProfCLI collect -a -g /usr/bin/stress-ng --cpu 8 --io 4 --vm 2 --vm-bytes 128M --fork 4 --timeout 20s
```
8.4.6 Constraints

- Do not move the downloaded kernel debug info from its default path.
- If the kernel version gets upgraded, then download the kernel debug info for the latest kernel version. AMD uProf would not show correct source and assembly if there is any mismatch between kernel debug info and kernel version.
- While profiling or analyzing kernel samples, do not reboot the system in between. Rebooting the system would cause the kernel to load at a different virtual address due to the KASLR feature of Linux kernel.
- The settings in the `/proc/sys/kernel/kptr_restrict` file enables AMD uProf to resolve kernel symbols and attribute samples to kernel functions. It does not enable the source and assembly level, call-graph analysis.

8.5 Holistic Analysis

The OS and runtime libraries can be traced along with CPU sampling-based profiles to provide timeline views to analyze what is happening in the system when that application is running. Following trace events can be collected and analyzed:

- OS scheduling event
- System calls
- POSIX threads library’s (pthread) thread synchronization APIs
- API and GPU activity tracing for heterogenous application using HIP
- MPI API event tracing

This tracing helps visualize:

- CPU thread states timeline – whether a thread is running, sleeping, or waiting for a resource to be available.
- Thread is running in kernel-space or user-space and if the thread is running in kernel-space, which system call is being called.
- If the thread is not scheduled to run, identify whether it is due to blocking I/O or waiting on synchronization object.
- If the profiled application is HIP based heterogeneous application, visualize the HIP & HSA APIs and GPU activities.
- MPI API timeline.

AMD uProf uses:

- Linux eBPF tracing framework to trace the OS and runtime events. Hence, OS Tracing option is supported only if eBPF is supported and BCC (BPF Compiler Collection) tool is installed on the host system. eBPF is supported in Linux kernel version 4.7 and later (it is recommended to use
kernel 4.15 or later). Use the command `AMDuProfCLI info --bpf` to check whether BPF support is available on the system and BCC is installed.

- AMD ROCtracer library to trace the ROCr (HSA API) and HIP supported GPU activity.
- MPI API tracing is supported using PMPI interface for MPICH and its derivatives.

**Supported Events**

Use the command `AMDuProfCLI info --list ostrace-events` to list the supported events on the target system. AMD uProf supports the following system events for tracing to show timeline view in GUI:

<table>
<thead>
<tr>
<th>Table 32. Supported Events for Holistic Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Category</strong></td>
</tr>
<tr>
<td>OS and Runtime</td>
</tr>
<tr>
<td>OS and Runtime</td>
</tr>
<tr>
<td>OS and Runtime</td>
</tr>
<tr>
<td>GPU</td>
</tr>
<tr>
<td>GPU</td>
</tr>
<tr>
<td>GPU</td>
</tr>
<tr>
<td>MPI</td>
</tr>
</tbody>
</table>

**Prerequisites**

For tracing OS events and runtime libraries:

- Requires Linux kernel 4.7 or later (it is recommended to use kernel 4.15 or later).
- Root access is required to trace the OS events in Linux.
- To install BCC and eBPF scripts, refer section “Installing BCC and eBPF” on page 21. To validate the BCC Installation, run the script `sudo AMDuProfVerifyBpfInstallation.sh`.

For tracing HIP, HSA APIs, and GPU activity — Requires AMD ROCm 4.30 to be installed. For the steps to install AMD ROCm, refer section “Installing ROCm” on page 21.

For tracing MPI APIs — MPICH 3.0 runtime or later
8.5.1 Holistic Analysis Using GUI

Configuring and Starting Profile

Complete the following steps to configure and start a profile:

1. Click **PROFILE > Start Profiling**.
   The Select Profile Target screen is displayed.
2. Select the appropriate profile target, click the **Next** button.
   The Select Profile Type screen is displayed.
3. From the Select Profile Type drop-down, select **CPU Profile** and then select any of the predefined sampling configurations.
4. Click **Advanced Options** to enable tracing options.
5. In **OS Tracing Options** pane, turn on the following tracing options as shown in the following figure:
   - Enable Context Switch trace
   - Enable System calls trace
   - Enable POSIX thread trace

![Figure 41. Advanced Options 1](image.png)
6. To enable GPU tracing for heterogeneous applications, in Advanced Options > GPU Tracing Options pane, turn on the required tracing option(s) as shown in the following figure:

![Advanced Options 2](image)

**Figure 42. Advanced Options 2**

7. Once all the options are set, click the Start Profile button at the bottom.

After the profile initialization the profile data collection screen is displayed.

**Analyzing Profile Data**

When the profiling stops, the collected raw profile data will be processed automatically and the Hot Spots screen of the Summary page is displayed.
Navigate to the TIMECHART page and select System Analysis in the Timeline Category dropdown. This view shows the timeline of combined OS and Runtime tracing and GPU activity tracing as follows:

Figure 43. System Analysis - TIMECHART

In the above figure:
1. The process or thread name along with the thread ID.
2. The OS and runtime characteristics for which tracing data is available for the current thread.
3. Timeline swim lanes for the various OS and runtime characteristics for which tracing data is available.

If huge number of samples are collected for an OS or runtime event, then selective samples (selected based on adaptive sampling approach) will be plotted. Either by increasing the zooming using Zoom Level slider or by performing selective filtering by selecting interesting time range in the Select Time Range to Focus axis will load more trace samples for analysis.

4. The GPU device(s) on which the kernels were running.
5. The GPU activities on this GPU device – GPU Copy and GPU Kernel executions.
6. To analyze the samples within the required time range, use Select Time Range to Focus axis to select the time-range so that, the GUI will only plot the trace samples within that time range.
7. To zoom in or zoom out, use the **Zoom Level** slider. The zoom in will fetch and render more trace samples.

### 8.5.2 Holistic Analysis Using CLI

The AMDuProfCLI can be used to collect the required trace data and generate the report in `.csv` format for further analysis. The processed profile data can also be imported in GUI.

**Collect Profile Data**

The CLI has an option called `--trace` to specify the OS events and runtime libraries to be traced.

Example CLI command to trace OS scheduling, system calls, and thread synchronization APIs along with time-based sampling for performing holistic analysis:

```
$ sudo AMDuProfCLI collect --config tbp --trace os -o /tmp/holistic-analysis/ /home/app/classic
```

This command will launch the program and collect the profile and trace data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile and trace data are saved.

In the above example, the session directory path is:

`/tmp/holistic-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27/`

For HIP based applications, example CLI command to trace OS scheduling, system calls, thread synchronization APIs, and GPU activity along with time-based sampling for performing holistic analysis:

```
$ sudo AMDuProfCLI collect --config tbp --trace os --trace gpu -o /tmp/holistic-analysis/ /home/app/classic
```

This command will launch the program and collect the profile and trace data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile and trace data are saved.

In the above example, the session directory path is:

`/tmp/holistic-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27/`

**Generate Profile Report**

Use the following CLI report command to generate the profile report in `.csv` format by passing the session directory path as the argument to `-i` option:

```
$ ./AMDuProfCLI report -i /tmp/holistic-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27 /
```

After processing the data and generating the report, the report file path is displayed on the terminal. An example of the OS trace report section in the .csv report file is as follows:

![OS Tracing Report Table]

Figure 44. System Analysis - OS Tracing Report

Analyze Trace Data with GUI

To visualize the trace data collected using CLI, the collected raw profile and trace data should be processed using CLI translate command and then it can be imported in the GUI. This will plot the timecharts and other views in the GUI using the profile and trace data collected.

Use the following CLI translate command invocation to process the raw trace records saved in the corresponding session directory path:

```
$ ./AMDuProfCLI translate -i /tmp/holistic-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27
...
Translation finished
```

Then import this session in the GUI by specifying the session directory path in Profile Data File text input box in the HOME > Import Session view. This will load the profile data saved in the session directory for further analysis.
8.6 Thread State Analysis

The OS and runtime libraries can be traced along with CPU sampling-based profiles to visualize the thread synchronization of a multi-threaded application and the threads core affinity. Following trace events should be collected for this analysis:

- OS scheduling event
- System calls
- POSIX threads library (pthread) thread synchronization APIs

This tracing helps visualize:

- CPU thread states timeline – whether a thread is running, sleeping, or waiting for a resource to be available.
- Thread is running in kernel-space or user-space and if the thread is running in kernel-space, the system call being called.
- If the thread is not scheduled to run, whether it is blocked due to I/O or waiting for a synchronization object.

Prerequisites

For tracing OS events and runtime libraries:

- Requires Linux kernel 4.7 or later (it is recommended to use kernel 4.15 or later).
- Root access is required to trace the OS events in Linux.
- To install BCC and eBPF scripts, refer section “Installing BCC and eBPF” on page 21. To validate the BCC Installation, run the script `sudo AMDuProfVerifyBpfInstallation.sh`.

Configuring and Starting Profile

Complete the following steps to configure and start a profile:

1. Click PROFILE > Start Profiling.
   - The Select Profile Target screen is displayed.
2. Select the appropriate profile target, click the Next button.
   - The Select Profile Type screen is displayed.
3. From the Select Profile Type drop-down, select CPU Profile and then select any of the predefined sampling configurations.
4. Click Advanced Options to enable tracing options.
5. In **OS Tracing Options** pane, turn on the following tracing options as shown in the following figure:
   - Enable Context Switch trace
   - Enable System calls trace
   - Enable POSIX thread trace

6. Once all the options are set, click the **Start Profile** button at the bottom.

   After the profile initialization the profile data collection screen is displayed.

### Analyzing Profile Data

When the profiling stops, the collected raw profile data will be processed automatically and the **Hot Spots** screen of the **Summary** page is displayed.

Navigate to the **TIMECHART** page and select **Thread State Analysis** in the **Timeline Category** drop-down. This view shows the timeline of combined OS and runtime libraries events traced as follows:

![Figure 45. Thread State Analysis - TIMECHART](image)

In the above figure:
1. The process or thread name along with the thread ID.
2. The OS and runtime characteristics for which tracing data is available for the current thread.
3. Timeline swim lanes for the various OS and runtime characteristics for which tracing data is available.

If huge number of samples are collected for an OS or runtime event, then selective samples (selected based on adaptive sampling approach) will be plotted. Either by increasing the zooming using **Zoom Level** slider or by performing selective filtering by selecting interesting time range in the **Select Time Range to Focus** axis will load more trace samples for analysis.

4. To analyze the samples within the required time range, use **Select Time Range to Focus** axis to select the time-range so that, the GUI will only plot the trace samples within that time range.

5. To zoom in or zoom out, use the **Zoom Level** slider. The zoom in will fetch and render more trace samples.

### 8.6.1 Thread State Analysis Using CLI

The AMDuProfCLI can be used to collect the required trace data and generate the report in **.csv** format for further analysis. The processed profile data can also be imported in GUI.

**Collect Profile Data**

The CLI has an option called **--trace** to specify the OS events and runtime libraries to be traced.

Example CLI command to trace OS scheduling, system calls, and POSIX thread synchronization APIs along with time-based sampling for performing holistic analysis:

```
$ sudo AMDuProfCLI collect --config tbp --trace os -o /tmp/holistic-analysis/ /home/app/classic
```

Generated data files path: /tmp/holistic-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27/

This command will launch the program and collect the profile and trace data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile and trace data are saved.

In the above example, the session directory path is:

```
/tmp/holistic-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27/
```

**Generate Profile Report**

Use the following CLI report command to generate the profile report in **.csv** format by passing the session directory path as the argument to **-i** option:

```
$ ./AMDuProfCLI report -i /tmp/thread-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27
```

After processing the data and generating the report, the report file path is displayed on the terminal. An example of the OS trace report section in the .csv report file is as follows:

![Figure 46. Thread State Analysis - OS Tracing Report](image)

**Analyze Trace Data with GUI**

To visualize the trace data collected using CLI, the collected raw profile and trace data should be processed using CLI translate command and then it can be imported in the GUI. This will plot the timecharts and other views in the GUI using the profile and trace data collected.

Use the following CLI translate command invocation to process the raw trace records saved in the corresponding session directory path:

```
$ ./AMDuProfCLI translate -i /tmp/thread-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27 ...
Translation finished
```

Then import this session in the GUI by specifying the session directory path in **Profile Data File** text input box in the **HOME > Import Session** view. This will load the profile data saved in the session directory for further analysis.
8.7 Kernel Block I/O Analysis

The Linux OS block I/O calls like insert, issue, and complete can be traced to provide the various metrics related to I/O operations performed by the application.

<table>
<thead>
<tr>
<th>Category</th>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS and Runtime</td>
<td>diskio</td>
<td>To trace the block I/O operations when the application is running.</td>
</tr>
</tbody>
</table>

This analysis can be used to analyze:

- Time taken to complete the I/O operations
- IOPS - Number of block I/O operations per second
- Read or Write bytes of block I/O operation
- Block I/O bandwidth

**Note:** The kernel can continue to perform the queued I/O requests submitted by the profiled application, even after the application exits. So, it is recommended to use system-wide tracing for this analysis.

**Prerequisites**

For tracing OS events and runtime libraries:

- Requires Linux kernel 4.7 or later (it is recommended to use kernel 4.15 or later).
- Root access is required to trace the OS events in Linux.
- To install BCC and eBPF scripts, refer section “Installing BCC and eBPF” on page 21. To validate the BCC Installation, run the script `sudo AMDuProfVerifyBpfInstallation.sh`.

8.7.1 Kernel Block I/O Analysis Using CLI

The AMDuProfCLI can be used to collect the required trace data and generate the report in `.csv` format for further analysis. The processed profile data can also be imported in GUI.

**Collect Profile Data**

Example CLI command to trace block I/O operations along with time-based sampling:

```
$ sudo AMDuProfCLI collect --config tbp --trace os=diskio -o /tmp/blockio-analysis/ /usr/bin/fio ...
...
Generated data files path: /tmp/blockio-analysis/AMDuProf-fio-OsTrace_Dec-09-2021_12-19-27
```

This command will launch the program and collect the profile and trace data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile and trace data are saved.
In the above example, the session directory path is:
/tmp/blockio-analysis/AMDuProf-fio-OsTrace_Dec-09-2021_12-19-27/

**Generate Profile Report**

Use the following CLI report command to generate the profile report in .csv format by passing the session directory path as the argument to -i option:

```
$ ./AMDuProfCLI report -i /tmp/blockio-analysis/AMDuProf-fio-OsTrace_Dec-09-2021_12-19-27
... Generated report file: /tmp/blockio-analysis/AMDuProf-fio-OsTrace_Dec-09-2021_12-19-27/report.csv
```

After processing the data and generating the report, the report file path is displayed on the terminal. An example of the disk I/O report section in the .csv report file is as follows:

![Disk I/O Summary Tables](image)

Figure 47. Disk I/O Summary Tables

**Analyze Trace Data with GUI**

To visualize the trace data collected using CLI, the collected raw profile and trace data should be processed using CLI translate command and then it can be imported in the GUI.

Use the following CLI translate command invocation to process the raw trace records saved in the corresponding session directory path:

```
$ ./AMDuProfCLI translate -i /tmp/blockio-analysis/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27
... Translation finished
```

Then import this session in the GUI by specifying the session directory path in Profile Data File text input box in the HOME > Import Session view. This will load the profile data saved in the session directory for further analysis.
8.7.2  Kernel Block I/O Analysis Using GUI

Configuring and Starting Profile
Complete the following steps to configure and start a profile:

1. Click **PROFILE > Start Profiling**.
   - The **Select Profile Target** screen is displayed.
2. Select the appropriate profile target, click the **Next** button.
   - The **Select Profile Type** screen is displayed.
3. From the **Select Profile Type** drop-down, select **CPU Profile** and then select any of the predefined sampling configurations.
4. Click **Advanced Options** to enable tracing options.
5. In **OS Tracing Options** pane, turn on the options **Enable Disk I/O trace**.
6. Once all the options are set, click the **Start Profile** button at the bottom.
   - After the profile initialization the profile data collection screen is displayed.

Analyzing Profile Data
When the profiling stops, the collected raw profile data will be processed automatically and the **Hot Spots** screen of the **Summary** page is displayed.

Navigate to the **ANALYZE** page and then select **Disk I/O Stats** in the vertical navigation bar as follows:

![ANALYZE - Block I/O Stats](Figure 48)

Figure 48. **ANALYZE - Block I/O Stats**
In the above figure:

1. The table shows various block I/O statistics at the device level.
2. The table shows various block I/O statistics of the I/O operations performed by an application on various devices.

### 8.8 GPU Offloading Analysis (GPU Tracing)

GPU offloading analysis is used to explore the traces of the function calls for a GPU compute-intensive application.

The AMD ROCtracer library provides support to capture the runtime APIs and GPU activities such as data transfer and kernel execution. This analysis helps to visualize the ROCr, HIP API calls, and GPU activities when a HIP based application is running. It is supported only with a launch application.

#### Supported Interfaces

AMD uProf supports tracing the following ROCr runtime APIs, GPU activities, and to show the data in GUI timeline view:

<table>
<thead>
<tr>
<th>Category</th>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>hip</td>
<td>HIP runtime trace</td>
</tr>
<tr>
<td>GPU</td>
<td>hsa</td>
<td>AMD ROCr runtime trace</td>
</tr>
<tr>
<td>GPU</td>
<td>kfd</td>
<td>AMD Kernel Fusion Driver(KFD) trace</td>
</tr>
</tbody>
</table>

#### Prerequisites

For tracing ROCr, HIP APIs, and GPU activities:

- Requires AMD ROCm 4.30 to be installed. For the steps to install AMD ROCm, refer section “Installing ROCm” on page 21.
- Support accelerators - AMD Instinct™ MI100 and MI200

### 8.8.1 GPU Offload Analysis Using GUI

#### Configuring and Starting Profile

Complete the following steps to configure and start a profile:

1. Click **PROFILE > Start Profiling**.
   
   The **Select Profile Target** screen is displayed.
2. Select the appropriate profile target, click the **Next** button.
   
   The **Select Profile Type** screen is displayed.
3. From the **Select Profile Type** drop-down, select **CPU Profile** and then select any of the predefined sampling configurations.

4. Click **Advanced Options** to enable tracing options.

5. In **GPU Tracing Options** pane, turn on the required options as follows:

![GPU Tracing Options](image)

**Figure 49. GPU Tracing Options**

6. Once all the options are set, click the **Start Profile** button at the bottom.

   After the profile initialization, the profile data collection screen is displayed.

**Analyzing Profile Data**

When the profiling stops, the collected raw profile data will be processed automatically and the **Hot Spots** screen of the **Summary** page is displayed.
Navigate to the TIMECHART page and then select GPU Offload Analysis in the Timeline Category drop-down. This view shows the timeline of ROCr, HIP runtime APIs tracing, and GPU activity tracing as follows:

![TIMECHART - GPU Offload Analysis](image)

**Figure 50.** TIMECHART - GPU Offload Analysis

In the above figure:

1. The process or thread name along with the thread ID.
2. The GPU runtime characteristics for which tracing data is available for the current thread.
3. Timeline swim lanes for the various GPU runtime APIs for which tracing data is available.
   
   If a large number of samples are collected for a trace event, then selective samples (selected based on adaptive sampling approach) will be plotted. Either by increasing the zooming using **Zoom Level** slider or by performing selective filtering by selecting interesting time range in the **Select Time Range to Focus** axis will load more trace samples for analysis.
4. The GPU device(s) on which the kernels were running.
5. The GPU activities on this GPU device – GPU Copy and GPU Kernel executions.
6. To analyze the samples within the required time range, use **Select Time Range to Focus** axis to select the time-range so that, the GUI will only plot the trace samples within that time range.
7. To zoom in or zoom out, use the **Zoom Level** slider. The zoom in will fetch and render more trace samples.
8.8.2 GPU Offload Analysis Using CLI

The AMDuProfCLI can be used to collect the required trace data and generate the report in .csv format for further analysis. The processed profile data can also be imported in GUI.

Collect Profile Data

The CLI has an option --trace to specify the GPU events and runtime libraries to be traced. For HIP based applications, example CLI command to trace ROCr, HIP APIs, and GPU activity along with time-based sampling for performing GPU offload analysis:

```bash
$ sudo AMDuProfCLI collect --config tbp --trace gpu -o /tmp/gpu-analysis/ /home/app/SampleApp

Generated data files path: /tmp/gpu-analysis/AMDuProf-SampleApp-GpuTrace_Dec-09-2021_12-19-27
```

This command will launch the program and collect the profile and trace data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile and trace data are saved.

In the above example, the session directory path is:

```
/tmp/gpu-analysis/AMDuProf-SampleApp-GpuTrace_Dec-09-2021_12-19-27/
```

Generate Profile Report

Use the following CLI report command to generate the profile report in .csv format by passing the session directory path as the argument to -i option:

```bash
$ ./AMDuProfCLI report -i /tmp/gpu-analysis/AMDuProf-SampleApp-GpuTrace_Dec-09-2021_12-19-27

```
After processing the data and generating the report, the report file path is displayed on the terminal. An example of the GPU trace report section in the `.csv` report file is as follows:

<table>
<thead>
<tr>
<th></th>
<th>Count</th>
<th>Elapsed Time (sec)</th>
<th>Avg</th>
<th>Elapsed Time (%) From Total API Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KERNEL SUMMARY</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>void nonbondedForceKernel&lt;true&gt;</td>
<td>375</td>
<td>0.860377</td>
<td>0.00229434</td>
<td>38.6226</td>
</tr>
<tr>
<td>void bondedForceKernel&lt;long long&gt;</td>
<td>501</td>
<td>0.665191</td>
<td>0.00120797</td>
<td>27.1872</td>
</tr>
<tr>
<td>void modifiedExclusionForcesKernel&lt;true&gt;</td>
<td>501</td>
<td>0.254039</td>
<td>0.000507064</td>
<td>11.4039</td>
</tr>
<tr>
<td>void nonbondedForceKernel&lt;true&gt;</td>
<td>75</td>
<td>0.162181</td>
<td>0.00216241</td>
<td>7.28035</td>
</tr>
<tr>
<td>void nonbondedForceKernel&lt;true&gt;</td>
<td>51</td>
<td>0.151271</td>
<td>0.00296609</td>
<td>6.79059</td>
</tr>
<tr>
<td><strong>DATA TRANSFER SUMMARY</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CopyDeviceToDevice</td>
<td>1935</td>
<td>0.340213</td>
<td>0.00175872</td>
<td>46.1724</td>
</tr>
<tr>
<td>CopyHostToDevice</td>
<td>1875</td>
<td>0.315207</td>
<td>0.00016811</td>
<td>42.7661</td>
</tr>
<tr>
<td>FillBuffer</td>
<td>4501</td>
<td>0.081114</td>
<td>1.80E-05</td>
<td>11.0039</td>
</tr>
<tr>
<td>WriteImage</td>
<td>4</td>
<td>0.00041440000</td>
<td>0.00013602</td>
<td>0.056225</td>
</tr>
<tr>
<td><strong>HIP API SUMMARY</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hipLaunchKernel</td>
<td>6333</td>
<td>0.341009</td>
<td>5.38E-05</td>
<td>25.4831</td>
</tr>
<tr>
<td>hipMemcpyToHost</td>
<td>4</td>
<td>0.27429</td>
<td>0.0685724</td>
<td>20.4973</td>
</tr>
<tr>
<td>hipEventSynchronize</td>
<td>527</td>
<td>0.174111</td>
<td>0.00033038</td>
<td>13.0111</td>
</tr>
<tr>
<td>hipStreamCreateWithPriority</td>
<td>3</td>
<td>0.122127</td>
<td>0.0407091</td>
<td>9.1264</td>
</tr>
<tr>
<td>hipStreamSynchronize</td>
<td>130</td>
<td>0.120697</td>
<td>0.000928435</td>
<td>9.01947</td>
</tr>
<tr>
<td><strong>HSA API SUMMARY</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hsa_signal_wait_saccquire</td>
<td>5257</td>
<td>0.399099</td>
<td>7.59E-05</td>
<td>17.359</td>
</tr>
<tr>
<td>hsa_queue_create</td>
<td>4</td>
<td>0.139842</td>
<td>0.0349604</td>
<td>6.08246</td>
</tr>
<tr>
<td>hsa_signal_load_relaxed</td>
<td>21730</td>
<td>0.126678</td>
<td>5.84E-06</td>
<td>5.5186</td>
</tr>
<tr>
<td>hsa_signal_silent_store_relaxed</td>
<td>16365</td>
<td>0.113986</td>
<td>6.97E-06</td>
<td>4.5578</td>
</tr>
<tr>
<td><strong>KFD API SUMMARY</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hsaKmfGetClockCounters</td>
<td>624076</td>
<td>0.812533</td>
<td>1.30E-06</td>
<td>66.0578</td>
</tr>
<tr>
<td>hsaKmfAllocMemory</td>
<td>142</td>
<td>0.150765</td>
<td>0.00106172</td>
<td>12.257</td>
</tr>
<tr>
<td>hsaKmfWaitOnEvent</td>
<td>1609</td>
<td>0.127358</td>
<td>7.92E-05</td>
<td>10.354</td>
</tr>
<tr>
<td>hsaKmfWaitOnMultipleSignals</td>
<td>1617</td>
<td>0.149034</td>
<td>7.72E-06</td>
<td>10.1561</td>
</tr>
</tbody>
</table>

![Figure 51. GPU Tracing Report](image-url)
Analyze Trace Data with GUI

To visualize the trace data collected using CLI, the collected raw profile and trace data should be processed using CLI translate command and then it can be imported in the GUI. This will plot the timecharts and other views in the GUI for the collected profile and trace categories.

Use the following CLI translate command invocation to process the raw trace records saved in the corresponding session directory path:

```
$ ./AMDuProfCLI translate -i /tmp/gpu-analysis/AMDuProf-SampleApp-GpuTrace_Dec-09-2021_12-19-27
Translation finished
```

Then import this session in the GUI by specifying the session directory path in Profile Data File text input box in the HOME > Import Session view. This will load the profile data saved in the session directory for further analysis.

8.9 GPU Profiling

The AMD ROCprofiler library provides support to monitor GPU hardware performance events when GPU kernels are dispatched and executed. The derived performance metrics are computed and reported in the CSV report. It is supported only with a launch application.

Prerequisites

For GPU performance profiling:

- Requires AMD ROCm 4.30 to be installed. For the steps to install AMD ROCm, refer section “Installing ROCm” on page 21
- Supported accelerators - AMD Instinct™ MI100 and MI200

Supported Events and Metrics

The following GPU performance metrics are supported. Run AMDuProfCLI info --list gpu-events command to list the supported events on the target system.

The following table shows the list of supported events:

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRBM_COUNT</td>
<td>GPU free running clock</td>
</tr>
<tr>
<td>GRBM_GUI_ACTIVE</td>
<td>GPU busy clock</td>
</tr>
<tr>
<td>SQ_WAVES</td>
<td>Count number of waves sent to SQs. (per-simd, emulated, global)</td>
</tr>
<tr>
<td>TCC_HIT_sum</td>
<td>Number of cache hits.</td>
</tr>
<tr>
<td>TCC_MISS_sum</td>
<td>Number of cache misses. UC reads count as misses.</td>
</tr>
<tr>
<td>SQ_INSTS_VALU</td>
<td>Number of VALU instructions issued. (per-simd, emulated)</td>
</tr>
<tr>
<td>SQ_INSTS_SALU</td>
<td>Number of SALU instructions issued. (per-simd, emulated)</td>
</tr>
</tbody>
</table>
Table 35. Supported Events for GPU Profiling

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQ_INSTS_SMEM</td>
<td>Number of SMEM instructions issued (per-simd, emulated)</td>
</tr>
<tr>
<td>SQ_INSTS_LDS</td>
<td>Number of LDS instructions issued (including FLAT) (per-simd, emulated)</td>
</tr>
<tr>
<td>SQ_INSTS_GDS</td>
<td>Number of GDS instructions issued (per-simd, emulated)</td>
</tr>
<tr>
<td>TCC_EA_RDREQ_sum</td>
<td>Number of TCC/EA read requests (either 32-byte or 64-byte)</td>
</tr>
<tr>
<td>TCC_EA_RDREQ_32B_sum</td>
<td>Number of 32-byte TCC/EA read requests</td>
</tr>
<tr>
<td>SQ_ACTIVE_INST_VALU</td>
<td>Number of cycles the SQ instruction arbiter is working on a VALU instruction (per-simd, nondeterministic)</td>
</tr>
<tr>
<td>SQ_THREAD_CYCLES_VALU</td>
<td>Number of thread-cycles used to execute VALU operations (per-simd)</td>
</tr>
<tr>
<td>TA_FLAT_READ_WAVEFRONTS_sum</td>
<td>Number of flat opcode reads processed by the TA</td>
</tr>
<tr>
<td>TA_FLAT_WRITE_WAVEFRONTS_sum</td>
<td>Number of flat opcode writes processed by the TA</td>
</tr>
</tbody>
</table>

The following table shows the list of supported metrics:

Table 36. Supported Metrics for GPU Profiling

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU_UTIL (%)</td>
<td>GPU utilization in percentage</td>
</tr>
<tr>
<td>VALU_UTIL (%)</td>
<td>VALU utilization in percentage</td>
</tr>
<tr>
<td>VALU_THREAD_DIVERGENCE (%)</td>
<td>Average VALU thread divergence in percentage</td>
</tr>
<tr>
<td>L2_CACHE_HIT_RATE (%)</td>
<td>Average L2 cache hit rate in percentage</td>
</tr>
<tr>
<td>VALU_INSTR (IPW)</td>
<td>Average number of VALU instructions per wave</td>
</tr>
<tr>
<td>SALU_INSTR (IPW)</td>
<td>Average number of SALU instructions per wave</td>
</tr>
<tr>
<td>SMEM_INSTR (IPW)</td>
<td>Average number of SMEM instructions per wave</td>
</tr>
<tr>
<td>LDS_INSTR (IPW)</td>
<td>Average number of LDS instructions per wave</td>
</tr>
<tr>
<td>GDS_INSTR (IPW)</td>
<td>Average number of GDS instructions per wave</td>
</tr>
<tr>
<td>L2_CACHE_HITS (PW)</td>
<td>Average number of L2 cache hits per wave</td>
</tr>
<tr>
<td>L2_CACHE_MISSES (PW)</td>
<td>Average number of L2 cache misses per wave</td>
</tr>
<tr>
<td>EA_32B_READ (PW)</td>
<td>Average number of 32-byte reads per wave</td>
</tr>
<tr>
<td>EA_64B_READ (PW)</td>
<td>Average number of 64-byte reads per wave</td>
</tr>
<tr>
<td>EA_READ_BW (GB/sec)</td>
<td>Read Bandwidth in GB per second</td>
</tr>
</tbody>
</table>
8.9.1 GPU Profiling Using CLI

Collect Profile Data

Use the following command to collect the GPU performance data:

```
$ sudo AMDuProfCLI collect --config gpu -o /tmp/ /home/app/SampleApp
```

Generated data files path: /tmp/AMDuProf-SampleApp-GPUProfile_Dec-09-2021_12-19-27

This command will launch the program and collect the profile data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile data are saved.

In the above example, the session directory path is:

/tmp/AMDuProf-SampleApp-GPUProfile_Dec-09-2021_12-19-27/

Generate Profile Report

Use the following CLI report command to generate the profile report in .csv format by passing the session directory path as the argument to -i option:

```
$ ./AMDuProfCLI report -i /tmp/AMDuProf-SampleApp-GPUProfile_Dec-09-2021_12-19-27
```

After processing the data and generating the report, the report file path is displayed on the terminal. An example of the GPU profile report section in the .csv report file is as follows:

**Figure 52. GPU Profile Report**

### 8.10 Other OS Tracing Events

Apart from the OS events that are listed in section “Holistic Analysis” on page 118, following OS events can also be traced along with CPU sampling-based profiles:

**Table 37. Supported Events for OS Tracing**

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagefault</td>
<td>To trace the number of page faults.</td>
</tr>
<tr>
<td>memtrace</td>
<td>To trace memory allocation and deallocation calls. By default, only memory allocations that are &gt;= 1KB are traced. <strong>Note:</strong> This is supported only for application level tracing.</td>
</tr>
</tbody>
</table>
Prerequisites

For tracing OS events and runtime libraries:

- Requires Linux kernel 4.7 or later (it is recommended to use kernel 4.15 or later).
- Root access is required to trace the OS events in Linux.
- To install BCC and eBPF scripts, refer section “Installing BCC and eBPF” on page 21. To validate the BCC Installation, run the script `sudo AMDuProfVerifyBpfInstallation.sh`.

8.10.1 Tracing Page Faults and Memory Allocations Using CLI

The AMDuProfCLI can be used to collect the required trace data and generate the report in `.csv` format for further analysis.

Collect Profile Data

The CLI has an option `--trace` to specify the OS events and runtime libraries to be traced. Example CLI command to trace page faults and memory allocations along with time-based sampling for performing holistic analysis:

```
$ sudo AMDuProfCLI collect --config tbp --trace os=pagefault,memtrace -o /tmp/ /home/app/classic
...
Generated data files path: /tmp/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27
```

This command will launch the program and collect the profile and trace data. Once the launched application is executed, the AMDuProfCLI will display the session directory path in which the raw profile and trace data are saved.

In the above example, the session directory path is:

```
/tmp/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27/
```

Generate Profile Report

Use the following CLI report command to generate the profile report in `.csv` format by passing the session directory path as the argument to `-i` option:

```
$ ./AMDuProfCLI report -i /tmp/AMDuProf-classic-OsTrace_Dec-09-2021_12-19-27
...
```
After processing the data and generating the report, the report file path is displayed on the terminal. An example of the GPU trace report section in the `.csv` report file is as follows:

<table>
<thead>
<tr>
<th>MONITORED EVENTS</th>
<th>Name</th>
<th>Threshold</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Trace Events:</td>
<td>PAGEFAULT</td>
<td></td>
<td>Page Faults for a process/thread</td>
</tr>
<tr>
<td></td>
<td>MEMTRACE</td>
<td>1024 bytes</td>
<td>Dynamic Memory Allocation tracing</td>
</tr>
</tbody>
</table>

**OS TRACING REPORT**

**PAGEFAULT SUMMARY**

<table>
<thead>
<tr>
<th>Process</th>
<th>Thread</th>
<th>User PF Count</th>
<th>Kernel PF Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>/home/amd/.../Scima</td>
<td>&quot;ScimarkStable(196941)&quot;</td>
<td>141</td>
<td>3</td>
</tr>
</tbody>
</table>

**MEMORY ALLOC SUMMARY**

<table>
<thead>
<tr>
<th>Process</th>
<th>Total Memory Allocated</th>
<th>Total Duration(seconds)</th>
<th>Memory Alloc</th>
<th>Memory Deallocate</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>/home/amd/.../Scima</td>
<td>0.097412</td>
<td>2.37E-05</td>
<td>7</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 53.  Monitored Events*

### 8.11 MPI Trace Analysis

MPI trace analysis can be used to analyze, compute, and message passing load imbalance among the ranks of a MPI application running on a cluster. It supports MPICH and derivative implementations. The supported thread models are FUNNLED and SERIALIZED. The profile reports are generated for Point-to-Point and Collective API activity summary.

**Support Matrix**

- MPI Spec versions: MPI-3.0
- Implementations: MPICH and derivatives
- Languages: C, C++, and Fortran

The AMDuProf CLI supports the following 2 modes for MPI tracing:

- **LWT** – Light-weight tracing is useful for quick analysis of an application. The report gets generated in `.csv` format on-the-fly during collection stage.
- **FULL** – Full tracing is useful for in-depth analysis. This mode requires post-processing for report generation in `.csv` format.
8.11.1 MPI Light-weight Tracing Using CLI

In LWT mode, quick report gets generated during collection stage. This mode supports limited set of APIs for tracing. This report gives overview of the application runtime activity as follows:

Table 38. List of Supported MPI APIs for Light-weight Tracing

<table>
<thead>
<tr>
<th>MPI_Bsend</th>
<th>MPI_Recv_init</th>
<th>MPI_Bcast</th>
<th>MPI_Ireduce_scatter</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Bsend_Init</td>
<td>MPI_Rsend</td>
<td>MPI_Gather</td>
<td>MPI_Iscan</td>
</tr>
<tr>
<td>MPI_Ibsend</td>
<td>MPI_Rsend_init</td>
<td>MPI_Gather_init</td>
<td>MPI_Iscatter</td>
</tr>
<tr>
<td>MPI_Improbe</td>
<td>MPI_Send</td>
<td>MPI_Iallgather</td>
<td>MPI_Iscatterv</td>
</tr>
<tr>
<td>MPI_Imrecv</td>
<td>MPI_Send_init</td>
<td>MPI_Iallgatherv</td>
<td>MPI_reduce</td>
</tr>
<tr>
<td>MPI_Iprobe</td>
<td>MPI_Ssend</td>
<td>MPI_Iallreduce</td>
<td>MPI_reduce_scatter</td>
</tr>
<tr>
<td>MPI_Irecv</td>
<td>MPI_Ssend_Init</td>
<td>MPI_Ialltoall</td>
<td>MPI_Scan</td>
</tr>
<tr>
<td>MPI_Irsend</td>
<td>MPI_Allgather</td>
<td>MPI_Ialltoallv</td>
<td>MPI_Scatter</td>
</tr>
<tr>
<td>MPI_Isend</td>
<td>MPI_Allgatherv</td>
<td>MPI_Ialltoallw</td>
<td>MPI_Scatterv</td>
</tr>
<tr>
<td>MPI_Issend</td>
<td>MPI_Allreduce</td>
<td>MPI_Ibarrier</td>
<td>MPI_Wait</td>
</tr>
<tr>
<td>MPI_Mprobe</td>
<td>MPI_Alltoall</td>
<td>MPI_Ibcast</td>
<td>MPI_Waitall</td>
</tr>
<tr>
<td>MPI_Mrecv</td>
<td>MPI_Alltoallv</td>
<td>MPI_Igather</td>
<td>MPI_Waitany</td>
</tr>
<tr>
<td>MPI_Probe</td>
<td>MPI_Alltoallw</td>
<td>MPI_Igatherv</td>
<td>MPI_Waitsome</td>
</tr>
<tr>
<td>MPI_Recv</td>
<td>MPI_BARRIER</td>
<td>MPI_Ireduce</td>
<td></td>
</tr>
</tbody>
</table>

Collect Profile Data

Example of a command to LWT trace an MPI application using AMDuProfCLI:

$ mpirun -np <number of processes> ./AMDuProfCLI collect --trace mpi=lwt -o <output_directory> <application>

After completing the tracing, the path to the session directory is displayed on the terminal. LWT report is generated immediately after completing the collection and saved into the session directory in: <output_directory>/<SESSION_DIR>/mpi/lwt/mpi-summary.csv.
An example of the LWT report section in the .csv file is as follows:

<table>
<thead>
<tr>
<th>Function</th>
<th>Min Time(seconds)</th>
<th>Max Time(seconds)</th>
<th>Average Time(seconds)</th>
<th>MPI Time(%)</th>
<th>Volume(Bytes)</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Probe</td>
<td>0</td>
<td>0.10431</td>
<td>0.01061</td>
<td>0.95905</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>MPI_Iprobe</td>
<td>0</td>
<td>0.04699</td>
<td>0</td>
<td>1.776</td>
<td>560933</td>
<td></td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>0.00003</td>
<td>0.06543</td>
<td>0.01478</td>
<td>3.5307</td>
<td>0</td>
<td>37</td>
</tr>
<tr>
<td>MPI_Barrier</td>
<td>0</td>
<td>0.0599</td>
<td>0.04337</td>
<td>17.91578</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>MPI_Recv</td>
<td>0</td>
<td>0.00135</td>
<td>0.00013</td>
<td>0.01193</td>
<td>899656</td>
<td>14</td>
</tr>
<tr>
<td>MPI_Irecv</td>
<td>0</td>
<td>0.00002</td>
<td>0.00001</td>
<td>0.00162</td>
<td>17046896</td>
<td>48</td>
</tr>
<tr>
<td>MPI_Send</td>
<td>0.00001</td>
<td>0.08166</td>
<td>0.02145</td>
<td>1.33841</td>
<td>899656</td>
<td>14</td>
</tr>
<tr>
<td>MPI_Isend</td>
<td>0</td>
<td>0.00038</td>
<td>0.00005</td>
<td>0.01567</td>
<td>17046896</td>
<td>48</td>
</tr>
<tr>
<td>MPI_Reduce</td>
<td>0</td>
<td>0.00014</td>
<td>0.00005</td>
<td>0.00815</td>
<td>14</td>
<td>24</td>
</tr>
<tr>
<td>MPI_Allreduce</td>
<td>0.00001</td>
<td>1.02037</td>
<td>0.09424</td>
<td>53.52716</td>
<td>3136</td>
<td>88</td>
</tr>
<tr>
<td>MPI_Bcast</td>
<td>0.00001</td>
<td>0.10316</td>
<td>0.04426</td>
<td>18.28283</td>
<td>60849496</td>
<td>64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rank</th>
<th>PID</th>
<th>MPI Time(seconds)</th>
<th>MPI Time(%)</th>
<th>Wait Time(seconds)</th>
<th>Call Count</th>
<th>Volume(Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>51907</td>
<td>0.77848</td>
<td>5.0246</td>
<td>0.01185</td>
<td>214</td>
<td>13075227</td>
</tr>
<tr>
<td>1</td>
<td>51902</td>
<td>2.21421</td>
<td>14.2914</td>
<td>0.07077</td>
<td>247754</td>
<td>11767163</td>
</tr>
<tr>
<td>2</td>
<td>51919</td>
<td>1.69282</td>
<td>10.92617</td>
<td>0.06905</td>
<td>196894</td>
<td>12326179</td>
</tr>
<tr>
<td>3</td>
<td>51917</td>
<td>2.57916</td>
<td>16.6499</td>
<td>0.11187</td>
<td>139521</td>
<td>11683555</td>
</tr>
<tr>
<td>4</td>
<td>51914</td>
<td>1.53032</td>
<td>9.87731</td>
<td>0.08567</td>
<td>106170</td>
<td>12317787</td>
</tr>
<tr>
<td>5</td>
<td>51906</td>
<td>2.32437</td>
<td>15.00242</td>
<td>0.04465</td>
<td>78779</td>
<td>11759619</td>
</tr>
<tr>
<td>6</td>
<td>51915</td>
<td>1.71808</td>
<td>11.08519</td>
<td>0.04018</td>
<td>36055</td>
<td>1230259</td>
</tr>
<tr>
<td>7</td>
<td>51905</td>
<td>2.65584</td>
<td>17.14191</td>
<td>0.113</td>
<td>217701</td>
<td>11677099</td>
</tr>
</tbody>
</table>

**Figure 54. LWT Report**

8.11.2 MPI Full Tracing Using CLI

Full tracing mode traces more APIs than LWT tracing. This mode is helpful for in-depth analysis of an MPI Application activity.

The report file for the full tracing includes multiple tables to represent various details:

- Communicator summary
- Rank summary
- P2P API summary
- Communication matrix
- Collective API summary
The list of supported MPI APIs is as follows:

### Table 39. MPI APIs

<table>
<thead>
<tr>
<th>MPI_API</th>
<th>MPI_API</th>
<th>MPI_API</th>
<th>MPI_API</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Pcontrol</td>
<td>MPI_Mrecv</td>
<td>MPI_Reduce</td>
<td>MPI_Iallreduce</td>
</tr>
<tr>
<td>MPI_Cancel</td>
<td>MPI_Irecv</td>
<td>MPI_Allreduce</td>
<td>MPI_Ialltoall</td>
</tr>
<tr>
<td>MPI_Probe</td>
<td>MPI_Send</td>
<td>MPI_Alltoall</td>
<td>MPI_IalltoallV</td>
</tr>
<tr>
<td>MPI_Iprobe</td>
<td>MPI_Bsend</td>
<td>MPI_AlltoallV</td>
<td>MPI_Ialltoallw</td>
</tr>
<tr>
<td>MPI_Mprobe</td>
<td>MPI_Ssend</td>
<td>MPI_Alltoallw</td>
<td>MPI_Ineighbor_Alltoall</td>
</tr>
<tr>
<td>MPI_Improbe</td>
<td>MPI_Rsend</td>
<td>MPI_Neighbor_Alltoall</td>
<td>MPI_Ineighbor_Alltoallw</td>
</tr>
<tr>
<td>MPI_Start</td>
<td>MPI_Bsend_init</td>
<td>MPI_Neighbor_Alltoallw</td>
<td>MPI_Ineighbor_Alltoallv</td>
</tr>
<tr>
<td>MPI_Startall</td>
<td>MPI_Ssend_init</td>
<td>MPI_Neighbor_Alltoallv</td>
<td>MPI_Ibarrier</td>
</tr>
<tr>
<td>MPI_Test</td>
<td>MPI_Rsend_init</td>
<td>MPI_Bcast</td>
<td>MPI_Ibcast</td>
</tr>
<tr>
<td>MPI_Testall</td>
<td>MPI_Send_init</td>
<td>MPI_Scan</td>
<td>MPI_Comm_create</td>
</tr>
<tr>
<td>MPI_Testany</td>
<td>MPI_Ibsend</td>
<td>MPI_Reduce_Scatter</td>
<td>MPI_Comm_dup</td>
</tr>
<tr>
<td>MPI_Testsome</td>
<td>MPI_Isend</td>
<td>MPI_Ireduce_Scatter</td>
<td>MPI_Comm_dup_with_info</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>MPI_Irsend</td>
<td>MPI_Iscan</td>
<td>MPI_Comm_split</td>
</tr>
<tr>
<td>MPI_Waitall</td>
<td>MPI_Isend</td>
<td>MPI_Iscatter</td>
<td>MPI_Comm_split_type</td>
</tr>
<tr>
<td>MPI_Waitany</td>
<td>MPI_Scatter</td>
<td>MPI_Iscatterv</td>
<td>MPI_Intercomm_create</td>
</tr>
<tr>
<td>MPI_Waitsome</td>
<td>MPI_Scatterv</td>
<td>MPI_Igather</td>
<td>MPI_Intercomm_merge</td>
</tr>
<tr>
<td>MPI_Barrier</td>
<td>MPI_Gather</td>
<td>MPI_Igatherv</td>
<td>MPI_Cart_create</td>
</tr>
<tr>
<td>MPI_Recv</td>
<td>MPI_Gatherv</td>
<td>MPI_Iallgather</td>
<td>MPI_Cart_sub</td>
</tr>
<tr>
<td>MPI_Irecv</td>
<td>MPI_Allgather</td>
<td>MPI_Iallgatherv</td>
<td>MPI_Graph_create</td>
</tr>
<tr>
<td>MPI_Sendrecv</td>
<td>MPI_Allgatherv</td>
<td>MPI_Ineighbor_Allgather</td>
<td>MPI_Dist_graph_create</td>
</tr>
<tr>
<td>MPI_Sendrecv_replace</td>
<td>MPI_Installer</td>
<td>MPI_Ineighbor_Allgather</td>
<td>MPI_Dist_graph_create_adjacent</td>
</tr>
<tr>
<td>MPI_Recv_Init</td>
<td>MPI_Installer</td>
<td>MPI_Ineighbor_Allgatherv</td>
<td>MPI_Ireduce</td>
</tr>
</tbody>
</table>

### Collect Profile Data

Example of a command to FULL trace an MPI application using AMD uProf CLI:

```sh
$ mpirun -np <number of processes> ./AMDuProfCLI collect --trace mpi=full -o <output_directory> <application>
```

After completing the tracing, the path to the session directory displayed on the terminal.
Generate Profile Report

Example of a command to generate the report in .csv format. Pass the session directory path with -i option:

```
$ ./AMDuProfCLI report -i <output_directory>/<SESSION_DIR>
```

After completing the report generation, the report.csv file path is displayed on the terminal.

Tables in the Report file

The following screenshots show example sections of a full tracing report file:

**Figure 55. MPI Communicator Summary Table**

![Table](image)

**Figure 56. MPI Rank Summary Table**

![Table](image)
Figure 57. MPI Point-to-Point API Summary Table

<table>
<thead>
<tr>
<th>Function</th>
<th>Min Time(seconds)</th>
<th>Max Time(seconds)</th>
<th>Average Time(seconds)</th>
<th>MPI Time(%)</th>
<th>Volume(Bytes)</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Wait</td>
<td>0.00000003</td>
<td>78.6133</td>
<td>0.00332891</td>
<td>88.07</td>
<td>0</td>
<td>1613760</td>
</tr>
<tr>
<td>MPI_Irecv</td>
<td>0.00000004</td>
<td>0.00148102</td>
<td>0.00000966</td>
<td>0.01</td>
<td>121926902688.00</td>
<td>806880</td>
</tr>
<tr>
<td>MPI_Isend</td>
<td>0.000000061</td>
<td>0.0011091</td>
<td>0.000002503</td>
<td>0.03</td>
<td>103659994656.00</td>
<td>806880</td>
</tr>
<tr>
<td>MPI_Comm_dup</td>
<td>0.000016531</td>
<td>0.000523422</td>
<td>0.000206985</td>
<td>0</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>MPI_Comm_split</td>
<td>0.0000139847</td>
<td>0.00040156</td>
<td>0.00028519</td>
<td>0</td>
<td>0</td>
<td>48</td>
</tr>
<tr>
<td>MPI_Cart_create</td>
<td>0.000022733</td>
<td>0.00062688</td>
<td>0.000295566</td>
<td>0</td>
<td>0</td>
<td>32</td>
</tr>
</tbody>
</table>

Figure 58. MPI Communication Matrix

<table>
<thead>
<tr>
<th>Rank --&gt; Rank</th>
<th>MPI Time(seconds)</th>
<th>MPI Time(%)</th>
<th>Volume(Bytes)</th>
<th>Volume(%)</th>
<th>Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 --&gt; 1</td>
<td>0.0369763</td>
<td>0</td>
<td>1772934048</td>
<td>0.73</td>
<td>16810</td>
</tr>
<tr>
<td>0 --&gt; 4</td>
<td>0.0734582</td>
<td>0</td>
<td>2481247344</td>
<td>1.02</td>
<td>16810</td>
</tr>
<tr>
<td>1 --&gt; 0</td>
<td>0.0289881</td>
<td>0</td>
<td>1772934048</td>
<td>0.73</td>
<td>16810</td>
</tr>
<tr>
<td>1 --&gt; 2</td>
<td>0.0301966</td>
<td>0</td>
<td>1772934048</td>
<td>0.73</td>
<td>16810</td>
</tr>
<tr>
<td>1 --&gt; 5</td>
<td>0.0565452</td>
<td>0</td>
<td>2540520192</td>
<td>1.04</td>
<td>16810</td>
</tr>
<tr>
<td>2 --&gt; 1</td>
<td>0.0343381</td>
<td>0</td>
<td>1772934048</td>
<td>0.73</td>
<td>16810</td>
</tr>
<tr>
<td>2 --&gt; 3</td>
<td>0.0328948</td>
<td>0</td>
<td>1772934048</td>
<td>0.73</td>
<td>16810</td>
</tr>
<tr>
<td>2 --&gt; 6</td>
<td>0.0581532</td>
<td>0</td>
<td>2540520192</td>
<td>1.04</td>
<td>16810</td>
</tr>
<tr>
<td>3 --&gt; 2</td>
<td>0.0353417</td>
<td>0</td>
<td>1772934048</td>
<td>0.73</td>
<td>16810</td>
</tr>
<tr>
<td>3 --&gt; 7</td>
<td>0.0633404</td>
<td>0</td>
<td>25040996160</td>
<td>1.03</td>
<td>16810</td>
</tr>
<tr>
<td>4 --&gt; 0</td>
<td>0.0363977</td>
<td>0</td>
<td>2481247344</td>
<td>1.02</td>
<td>16810</td>
</tr>
<tr>
<td>4 --&gt; 5</td>
<td>0.0274887</td>
<td>0</td>
<td>1832206896</td>
<td>0.75</td>
<td>16810</td>
</tr>
<tr>
<td>4 --&gt; 8</td>
<td>0.0665892</td>
<td>0</td>
<td>2481247344</td>
<td>1.02</td>
<td>16810</td>
</tr>
<tr>
<td>5 --&gt; 1</td>
<td>0.0326747</td>
<td>0</td>
<td>2540520192</td>
<td>1.04</td>
<td>16810</td>
</tr>
<tr>
<td>5 --&gt; 4</td>
<td>0.0225724</td>
<td>0</td>
<td>1832206896</td>
<td>0.75</td>
<td>16810</td>
</tr>
<tr>
<td>5 --&gt; 6</td>
<td>0.029573</td>
<td>0</td>
<td>1832206896</td>
<td>0.75</td>
<td>16810</td>
</tr>
<tr>
<td>5 --&gt; 9</td>
<td>0.0645192</td>
<td>0</td>
<td>2540520192</td>
<td>1.04</td>
<td>16810</td>
</tr>
<tr>
<td>6 --&gt; 2</td>
<td>0.0428896</td>
<td>0</td>
<td>2540520192</td>
<td>1.04</td>
<td>16810</td>
</tr>
<tr>
<td>6 --&gt; 5</td>
<td>0.0323074</td>
<td>0</td>
<td>1832206896</td>
<td>0.75</td>
<td>16810</td>
</tr>
<tr>
<td>6 --&gt; 7</td>
<td>0.0325295</td>
<td>0</td>
<td>1832206896</td>
<td>0.75</td>
<td>16810</td>
</tr>
</tbody>
</table>

Figure 59. MPI Collective API Summary Table

<table>
<thead>
<tr>
<th>Function</th>
<th>Min Time(seconds)</th>
<th>Max Time(seconds)</th>
<th>Average Time(seconds)</th>
<th>MPI Time(%)</th>
<th>Input Volume(Bytes)</th>
<th>Output Volume(Bytes)</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Scatterv</td>
<td>0.0000000049</td>
<td>0.110154</td>
<td>0.000033863</td>
<td>0.32</td>
<td>493951444</td>
<td>493951444</td>
<td>573760</td>
</tr>
<tr>
<td>MPI_Gather</td>
<td>0.00000016</td>
<td>3.67246</td>
<td>0.000763753</td>
<td>0.73</td>
<td>233984</td>
<td>233984</td>
<td>58496</td>
</tr>
<tr>
<td>MPI_Gatherv</td>
<td>0.00000014</td>
<td>1.72094</td>
<td>0.00146369</td>
<td>1.17</td>
<td>5855507368</td>
<td>5855507368</td>
<td>48704</td>
</tr>
<tr>
<td>MPI_Alldistribute</td>
<td>0.000004759</td>
<td>0.0151899</td>
<td>0.000302453</td>
<td>0</td>
<td>448</td>
<td>448</td>
<td>80</td>
</tr>
<tr>
<td>MPI_Bcast</td>
<td>0.000000231</td>
<td>5.48656</td>
<td>0.00504196</td>
<td>9.66</td>
<td>5125009020</td>
<td>341667268</td>
<td>116880</td>
</tr>
</tbody>
</table>
8.11.3 MPI FULL Tracing Using GUI

Collecting and Importing a Trace

Use CLI to trace a target MPI application and generate the report using CLI. For the steps, refer section “MPI Full Tracing Using CLI” on page 144. Import the report to GUI as shown in the following figure to analyze the trace data:

![Import Profile Session](image)

**Figure 60. Import Profile Session**

Analyzing the Profile Data

MPI Communication Matrix

After the import is complete, use **MPI Communication Matrix** view to analyze the MPI trace data in the GUI. Navigate to HPC > MPI Flat Profile to view the MPI communication matrix visualizer. This view displays rank-to-rank communication summary in matrix format.

Following figure shows the MPI communication matrix:

![MPI Communication Matrix](image)

**Figure 61. MPI Communication Matrix**
In the above figure:
1. Ranks ordered in row-wise and column-wise.
2. Each cell displays the total data volume transferred from one rank to another rank.
3. Tool-tip shows additional details when the mouse is hovered over a cell.
5. Sum of all the data transfers for the rank.
6. Mean of all the data transfers for the rank.

MPI Rank Analysis View

Navigate to **TIMECHART > MPI Rank Analysis** in the **Timeline Category** drop-down. This view displays MPI API calling activities in the timeline graph as follows:

![Figure 62. TIMECHART - MPI Rank Analysis](image)

In the above screenshot:
1. Rank ID
2. MPI API activity
3. Tool-tip shows more info about the MPI API activity
4. To select a specific time-range
5. To zoom in/out the visualizer
Chapter 9  Power Profile

System-wide Power Profile

The AMD uProf profiler offers live power profiling to monitor the behavior of the systems based on AMD CPUs, APUs, and discrete GPUs. It provides various counters to monitor power and thermal characteristics.

These counters are collected from various resources such as RAPL, SMU, and MSRs. They are collected at regular time interval and either reported as a text file or plotted as line graphs. They can also be saved into the database for future analysis.

Features

AMD uProf comprises of the following features:

• The GUI can be used to configure and monitor the supported power metrics.

• The TIMECHART page helps to monitor and analyze:
  – Logical Core level metrics – Core Effective Frequency and P-State
  – Physical Core level metrics – RAPL based Core Power and Temperature
  – Package level metrics – RAPL based Package Power
  – GPU metrics – power, temperature, and frequency

• AMDuProfCLI’s timechart command collects the system metrics and writes into a text file or comma-separated-value (CSV) file.

• AMDPowerProfileApi library provides APIs to configure and collect the supported system level performance, thermal and power metrics of AMD CPU/APUs and dGPUs.

• The collected live profile data can be stored in the database for future analysis.

9.1  Metrics

The supported metrics depend on the processor family and model and are broadly grouped under various categories. Following are the supported counter categories by processor families:

Table 40. Family 17h Model 00h – 0Fh (AMD Ryzen™, AMD Ryzen ThreadRipper™, and 1st Gen AMD EPYC™)

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on the platform activity levels. It is available for Core and Package.</td>
</tr>
<tr>
<td>Frequency</td>
<td>Core Effective Frequency for the sampling period, reported in MHz.</td>
</tr>
<tr>
<td>Temperature</td>
<td>Average temperature for the sampling period, reported in Celsius. The temperature reported is with reference to Tctl. It is available for Package.</td>
</tr>
</tbody>
</table>
### Table 40. Family 17h Model 00h – 0Fh (AMD Ryzen™, AMD Ryzen ThreadRipper™, and 1st Gen AMD EPYC™)

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-State</td>
<td>CPU Core P-State at the time when sampling was performed.</td>
</tr>
</tbody>
</table>

### Table 41. Family 17h Model 10h — (AMD Ryzen™ and AMD Ryzen™ PRO APU)

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Available for Core and Package.</td>
</tr>
<tr>
<td>Frequency</td>
<td>Core Effective Frequency for the sampling period, reported in MHz</td>
</tr>
<tr>
<td>Temperature</td>
<td>Average temperature for the sampling period, reported in Celsius. Temperature reported is with reference to Tctl. Available for Package.</td>
</tr>
<tr>
<td>P-State</td>
<td>CPU Core P-State at the time when sampling was performed.</td>
</tr>
</tbody>
</table>

### Table 42. Family 17h Model 70h — 7Fh (3rd Gen AMD Ryzen™)

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Available for Core and Package.</td>
</tr>
<tr>
<td>Frequency</td>
<td>Core Effective Frequency for the sampling period, reported in MHz</td>
</tr>
<tr>
<td>P-State</td>
<td>CPU Core P-State at the time when sampling was performed.</td>
</tr>
<tr>
<td>Temperature</td>
<td>Average temperature for the sampling period, reported in Celsius. Temperature reported is with reference to Tctl. Available for Package.</td>
</tr>
</tbody>
</table>

### Table 43. Family 17h Model 30h — 3Fh (EPIC 7002)

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Available for Core and Package.</td>
</tr>
<tr>
<td>Frequency</td>
<td>Core Effective Frequency for the sampling period, reported in MHz</td>
</tr>
<tr>
<td>P-State</td>
<td>CPU Core P-State at the time when sampling was performed.</td>
</tr>
<tr>
<td>Temperature</td>
<td>Average temperature for the sampling period, reported in Celsius. Temperature reported is with reference to Tctl. Available for Package.</td>
</tr>
</tbody>
</table>
Supported Counter categories for older APU families

### Table 44. Supported Counter Categories for Older APU Families

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Available for APU, ComputeUnit, iGPU, PCIe Controller, Memory Controller, Display Controller, and VDDCR_SOC.</td>
</tr>
<tr>
<td>Frequency</td>
<td>Effective Frequency for the sampling period, reported in MHz. Available for Core and iGPU.</td>
</tr>
<tr>
<td>Temperature</td>
<td>Average estimated temperature for the sampling period, reported in Celsius. Calculated based on socket activity levels, normalized, and scaled, relative to the specific processor's maximum operating temperature. Available for CPU ComputeUnit and iGPU.</td>
</tr>
<tr>
<td>P-State</td>
<td>CPU Core P-State at the time when sampling was performed.</td>
</tr>
<tr>
<td>Controllers</td>
<td>Socket PPT Limit and Power.</td>
</tr>
<tr>
<td>CorrelatedPower</td>
<td>Correlated Average Power for the sampling period, reported in Watts. This is an estimated consumption value based on platform activity levels. Available for APU, CPU ComputeUnit, VDDGFX, VDDIO, VDDNB, VDDP, UVD, VCE, ACP, UNB, SMU, and RoC.</td>
</tr>
</tbody>
</table>

### Table 45. Supported Categories for dGPUs

<table>
<thead>
<tr>
<th>Power Counter Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power</td>
<td>Average estimated dGPU power for the sampling period, reported in Watts. Calculated based on dGPU activity levels.</td>
</tr>
<tr>
<td>Frequency</td>
<td>Average dGPU frequency for the sampling period, reported in MHz.</td>
</tr>
<tr>
<td>Temperature</td>
<td>Average estimated dGPU temperature for the sampling period, reported in Celsius.</td>
</tr>
<tr>
<td>Voltage</td>
<td>CPU Core P-State at the time when sampling was performed.</td>
</tr>
<tr>
<td>Current</td>
<td>Socket PPT Limit and Power.</td>
</tr>
</tbody>
</table>

### 9.2 Using Profile through GUI

#### System-wide Power Profile (Live)

This profile type is used to perform the power analysis where the metrics are plotted in a live timeline graph and/or saved in a database. Complete the following steps to configure and start the profile:
9.2.1 Configuring a Profile

Complete the following steps to configure a profile:

1. Click the **PROFILE** tab at the top navigation bar or one of the following on the *Welcome* page:
   - Profile entire System
   - See What’s guzzling power in your system
   
   The *Select Profile Target* page is displayed.

2. Click the **Next** button.

   The *Select Profile Type* page is displayed.

3. From the *Select Profile Type* drop-down, select the option **System-wide Power Profile (Live)**.

   All the live profiling options and available counters are displayed in the respective panes as follows:

   ![Live System-wide Power Profile](image)

   **Figure 63.** Live System-wide Power Profile (from AMD uProf 3.4)

4. In the *Counters* pane, select the required counter category and the respective options.

   **Note:** You can configure multiple counter categories.

   During the profiling, you can render the graphs live.
5. Click the **Start Profile** button.

In this profile type, the profile data will be generated as line graphs in the **TIMECHART** page for further analysis.

### 9.2.2 Analyzing a Profile

Once the required counters are selected and the profile data collection begins, the **TIMECHART** tab will open and the metrics will be plotted in the live timeline graphs.

![Image of Timechart Page]

**Figure 64. Timechart Page**

1. In the **TIMECHART** page, the metrics will be plotted in the live timeline graphs. The line graphs are grouped together and plotted based on the category.

2. There is a data table adjacent to each graph to display the current value of the counters.

3. From the **Graph Visibility** pane, you can choose the graph to display.

4. When plotting is in progress, you can:
   - Click the **Pause Graphs** button to pause the graphs without pausing the data collection. You can click the **Play Graphs** button to resume them later.
   - Click the **Stop Profiling** button to stop the profiling without closing the view. This will stop collecting the profile data.
   - Click the **Close View** button to stop the profiling and close the view.
9.3 Using CLI to Profile

You can use AMDuProfCLI’s timechart command to collect the system metrics and write them into a text file or comma-separated-value (CSV) file. To collect power profile counter values, complete the following steps:

1. Run the command with `--list` option to get the list of supported counter categories.
2. Use the command to specify the required counters with `-e` or `--event` option to collect and report the required counters.

The timechart run to list the supported counter categories is as follows:

![Figure 65. --list Command Output]

The above run will collect the power and frequency counters on all the devices on which these counters are supported and writes them in the output file specified with `-o` option. Before the profiling
begins, the given application will be launched and the data will be collected till the application terminates.

### 9.3.1 Examples

#### Windows

- Collect all the power counter values for a duration of 10 seconds with a sampling interval of 100 milliseconds:
  ```shell
c:\> AMDuProfCLI.exe timechart --event power --interval 100 --duration 10
  ```

- Collect all frequency counter values for 10 seconds, sampling them every 500 milliseconds and adding the results to a csv file:
  ```shell
c:\> AMDuProfCLI.exe timechart --event frequency -o C:\Temp\Poweroutput --interval 500 --duration 10
  ```

- Collect all the frequency counter values at core 0 to 3 for 10 seconds, sampling them every 500 milliseconds and adding the results to a text file:
  ```shell
c:\> AMDuProfCLI.exe timechart --event core=0-3,frequency -o C:\Temp\Poweroutput --interval 500 --duration 10 --format txt
  ```

#### Linux

- Collect all the power counter values for a duration of 10 seconds with a sampling interval of 100 milliseconds:
  ```shell
$ ./AMDuProfCLI timechart --event power --interval 100 --duration 10
  ```

- Collect all the frequency counter values for 10 seconds, sampling them every 500 milliseconds and adding the results to a csv file:
  ```shell
$ ./AMDuProfCLI timechart --event frequency -o /tmp/PowerOutput --interval 500 --duration 10
  ```

- Collect all the frequency counter values at core 0 to 3 for 10 seconds, sampling them every 500 milliseconds and adding the results to a text file:
  ```shell
$ ./AMDuProfCLI timechart --event core=0-3,frequency -o /tmp/PowerOutput --interval 500 --duration 10 --format txt
  ```

### 9.4 AMDPowerProfileAPI Library

AMDPowerProfileAPI library provides APIs to configure and collect the supported power profiling counters on various AMD platforms directly without using AMD uProf GUI or CLI. The AMDPowerProfileAPI library is used to analyze the power efficiency of systems based on AMD CPUs, APUs, and dGPUs (Discrete GPU).

These APIs provide interface to read the power, thermal, and frequency characteristics of AMD APUs, dGPUs, and their subcomponents. These APIs are targeted for software developers who want to write their own application to sample the power counters based on their specific use case(s).
For a detailed information on these APIs, refer AMDPowerProfilerAPI.pdf in the AMD uProf installation folder.

### 9.4.1 Using the APIs

Refer the sample program `CollectAllCounters.cpp` on how to use these APIs. The program must be linked with the AMDPowerProfileAPI library while compiling. The power profiling driver must be installed and running.

A sample program `CollectAllCounters.cpp` that uses these APIs is available at `<AMDuProf-install-dir>/Examples/CollectAllCounters/ dir`. To build and execute the sample application, complete the following steps based on the OS that you are using:

**Windows**

A Visual Studio 2015 solution file `CollectAllCounters.sln` is available at `/C:/Program Files/AMD/AMDuProf/Examples/CollectAllCounters/ folder` to build the sample program.

**Linux**

1. Execute the following commands to build:

   ```bash
   $ cd <AMDuProf-install-dir>/Examples/CollectAllCounters
   $ g++ -O -std=c++11 CollectAllCounters.cpp -I<AMDuProf-install-dir>/include -lAMDPowerProfileAPI -L<AMDuProf-install-dir>/bin -Wl,-rpath <AMDuProf-install-dir>/bin -o CollectAllCounters
   ```

2. Run the following commands to execute:

   ```bash
   $ export LD_LIBRARY_PATH=<AMDuProf-install-dir>/bin
   $ ./CollectAllCounters
   ```

### 9.5 Limitations

- Only one power profile session can run at a time.
- Minimum supported sampling period in CLI is 100ms. It is recommended to use a large sampling period to reduce the sampling and rendering overhead.
- Ensure that the latest Radeon driver is installed before running the power profiler. Newer version of dGPU may go to sleep (low power) state frequently if there is no activity in dGPU. In that case, the power profiler may display a warning `AMDT_WARN_SMU_DISABLED`. The counters may not be accessible in this state. Before running the power profiler, it is advisable to bring the dGPU to an active state.
- ICELAND dGPU series (Topaz-XT, Topaz PRO, Topaz XTL, and Topaz LE) is not supported.
- If SMU becomes inaccessible while profiling is in progress, the behavior will be undefined.
Chapter 10  Energy Analysis

Power Application Analysis

AMD uProf profiler offers Power Application Analysis to identify energy hotspots in the application. This is Windows OS only functionality. This profile type is used to analyze the energy consumption of an application or processes running in the system.

Features

- Profile data — Periodically RAPL core energy values are sampled using OS timer as sampling event
- Profile mode — Profile data is collected when the application is running in user and kernel mode
- Profiles
  - C, C++, FORTRAN, and assembly applications
  - Various software components – Applications, dynamically linked/loaded modules, and OS kernel modules
- Profile data is attributed at various granularities
  - Process, Thread, Load Module, Function, or Source line
  - To correlate the profile data to Function and Source line, the debug information generated by the compiler is required
- Processed profile data is stored in databases, which can be used to generate reports later
- Profile reports are available in comma-separated-value (CSV) format to use with spreadsheets
- Hot spots summary
- Process and function analysis
- Source and disassembly analysis

10.1 Using CLI to Profile

To profile and analyze the performance of a native (C/C++) application, complete the following steps:

1. Prepare the application. For more information, refer “Reference” on page 163.
2. Collect the samples for the application using AMDuProfCLI’s collect command.
3. Generate the report using AMDuProfCLI’s report command in a readable format for analysis.

Preparing the application is to build the launch application with debug information as debug info is needed to correlate the samples to functions and source lines.

The collect command will launch the application (if given), collect the profile data, and generate raw data file (.pdata on Windows) and other miscellaneous files.
The report command translates the collected raw profile data to aggregate and attribute to the respective processes, threads, load modules, functions, and instructions. Then it writes them into a database and generates a report in CSV format.

**Example**

- Launch `classic.exe` and collect energy samples for that launch application:
  ```
  C:\> AMDuProfCLI.exe collect --config energy -o c:\Temp\pwrprof classic.exe
  ```

- Generate a report from the session data folder:
  ```
  C:\> AMDuProfCLI.exe report -i c:\Temp\pwrprof\<session-dir>
  ```

- Generate a report from raw .pdata file and use Symbol Server paths to resolve symbols:
  ```
  C:\> AMDuProfCLI.exe report --symbol-path C:\AppSymbols;C:\DriverSymbols
  --symbol-server http://msdl.microsoft.com/download/symbols
  --cache-dir C:\symbols -i c:\Temp\pwrprof\<session-dir>
  ```

**10.2 Limitations**

- Only one energy analysis profile session can run at a time.
- This feature is only for the Windows OS.
Chapter 11  Profile Control APIs

11.1  AMDProfileControl APIs

The AMDProfileControl APIs allow you to limit the profiling scope to a specific portion of the code within the target application.

Usually, while profiling an application, samples for the entire control flow of the application execution will be collected, that is, from the start of execution till end of the application execution. The control APIs can be used to enable the profiler to collect data only for a specific part of application, for example, a CPU intensive loop and a hot function.

The target application needs to be recompiled after instrumenting the application to enable/disable profiling of the required code regions only.

Header Files

The application should include the header file `AMDProfileController.h` which declares the required APIs. This file is available in the include directory under AMD uProf’s install path.

Static Library

The instrumented application should link with the AMDProfileController static library available in:

**Windows**

`<AMDuProf-install-dir>\lib\x86\AMDProfileController.lib`
`<AMDuProf-install-dir>\lib\x64\AMDProfileController.lib`

**Linux**

`<AMDuProf-install-dir>/lib/x64/libAMDProfileController.a`

11.1.1  Profile Control APIs

These profile control APIs are available to pause and resume the profile data collection.

**amdProfileResume**

When the instrumented target application is launched through AMDuProf/AMDuProfCLI, the profiling will be in the paused state and no profile data will be collected till the application calls this resume API.

```c
bool amdProfileResume (AMD_PROFILE_CPU);
```

**amdProfilePause**

When the instrumented target application has to pause the profile data collection, this API must be called:

```c
bool amdProfilePause (AMD_PROFILE_CPU);
```
These APIs can be called multiple times within the application. Nested Resume - Pause calls are not supported. AMD uProf profiles the code within each Resume-Pause APIs pair. After adding these APIs, the target application should be compiled before initiating a profile session.

### 11.1.2 Using the APIs

Include the header file `AMDPProfileController.h` and call the resume and pause APIs within the code. The code encapsulated within resume-pause API pair will be profiled by the CPU Profiler.

These APIs can be:
- Called multiple times to profile different parts of the code.
- Spread across multiple functions, that is, resume called from one function and stop called from another function.
- Spread across threads, that is, resume called from one thread and stop called from another thread of the same target application.

In the following code snippet, the CPU Profiling data collection is restricted to the execution of `multiply_matrices()` function:

```c
#include <AMDPProfileController.h>

int main (int argc, char* argv[])
{
    // Initialize the matrices
    initialize_matrices ();

    // Resume the CPU profile data collection
    amdProfileResume (AMD_PROFILE_CPU);

    // Multiply the matrices
    multiply_matrices ();

    // Stop the CPU Profile data collection
    amdProfilePause (AMD_PROFILE_CPU);

    return 0;
}
```
11.1.3 Compiling Instrumented Target Application

Windows

To compile the application on Microsoft Visual Studio, update the configuration properties to include the path of header file and link it with *AMDProfileController.lib* library.

Linux

To compile a C++ application on Linux using g++, use the following command:

```bash
$ g++ -std=c++11 <sourcefile.cpp> -I <AMDuProf-install-dir>/include -L<AMDuProf-install-dir>/lib/x64/ -lAMDProfileController -lrt -pthread
```

*Note:* Do not use the *-static option while compiling with g++.*

11.1.4 Profiling Instrumented Target Application

**AMD uProf GUI**

After compiling the target application, create a profile configuration in AMD uProf, set the desired CPU profile session options. While setting the CPU profile session options, in the *Profile Scheduling* section, select *Are you using Profile Instrumentation API?*.

Once all the settings are done, start the CPU profiling. The profiling will begin in the paused state and the target application execution begins. When the resume API is called from target application, CPU Profile starts profiling till pause API is called from the target application or the application is terminated. When the pause API is called in the target application, the profiler stops profiling and waits for the next control API call.

**AMDuProfCLI**

To profile from CLI, option *--start-paused* should be used to start the profiler in a paused state.

**Windows**

```bash
C:\> AMDuProfCLI.exe collect --config tbp --start-paused -o C:\Temp\prof-tbp ClassicCpuProfileCtrl.exe
```

**Linux**

```bash
$ ./AMDuProfCLI collect --config tbp --start-paused -o /tmp/cpuprof-tbp /tmp/AMDuProf/Examples/ClassicCpuProfileCtrl1/ClassicCpuProfileCtrl1
```
Chapter 12  Reference

12.1  Preparing an Application for Profiling

The AMD uProf uses the debug information generated by the compiler to show the correct function names in various analysis views and to correlate the collected samples to source statements in Source page. Otherwise, the results of the CPU Profiler would be less descriptive, displaying only the assembly code.

12.1.1  Generating Debug Information on Windows

When using Microsoft Visual C++ to compile the application in release mode, set the following options before compiling the application to ensure that the debug information is generated and saved in a program database file (with a .pdb extension). To set the compiler option to generate the debug information for a x64 application in release mode, complete the following steps:

1. Right-click the project and select Properties from the menu.
2. From the Configuration drop-down, select Active(Release).
3. From the Platform drop-down, select Active(Win32) or Active(x64).
4. In the project pane on the left, expand Configuration Properties.
5. Expand C/C++ and select General.
6. In the work pane, select Debug Information Format.
7. From the drop-down, select **Program Database (/Zi)** or **Program Database for Edit & Continue (/ZI)**.

8. In the project pane, expand **Linker** and then select **Debugging**.

9. From the **Generate Debug Info** drop-down, select **/DEBUG**.

### 12.1.2 Generating Debug Information on Linux

The application must be compiled with the `-g` option to enable the compiler to generate debug information. Modify either the Makefile or the respective build scripts accordingly.

### 12.2 CPU Profiling

The AMD uProf CPU Performance Profiling follows a sampling-based approach to gather the profile data periodically. It uses a variety of software and hardware resources available in AMD x86 based processor families. CPU Profiling uses the OS timer, HW Performance Monitor Counters (PMC), and HW IBS feature.

The following section explains the various key concepts related to CPU Profiling.
12.2.1 Hardware Sources

Performance Monitor Counters (PMC)

AMD’s x86-based processors have Performance Monitor Counters (PMC) that help monitor various micro-architectural events in a CPU core. The PMC counters are used in two modes:

- Counting mode: These counters are used to count the specific events that occur in a CPU core.
- Sampling mode: These counters are programmed to count the specific number of events. Once the count reaches the appropriate number of times (called sampling interval), an interrupt is triggered. During the interrupt handling, the CPU Profiler collects the profile data.

The number of hardware performance event counters available in each processor is implementation-dependent (refer to the BIOS and Kernel Developer’s Guide [BKDG] of the specific processor for the exact number of hardware performance counters). The operating system and/or BIOS can reserve one or more counters for internal use. Thus, the actual number of available hardware counters may be less than the number of hardware counters. The CPU Profiler uses all available counters for profiling.

Instruction-Based Sampling (IBS)

IBS is a code profiling mechanism that enables the processor to select a random instruction fetch or micro-Op after a programmed time interval has expired and record specific performance information about the operation. An interrupt is generated when the operation is complete as specified by IBS Control MSR. An interrupt handler can then read the performance information that was logged for the operation.

The IBS mechanism is split into two parts:

- Instruction Fetch performance
- Instruction Execution Performance

The instruction fetch sampling provides information about instruction TLB and instruction cache behavior for fetched instructions.

Instruction execution sampling provides information about micro-Op execution behavior.

The data collected for the instruction fetch performance is independent of the data collected for the instruction execution performance. The support for the IBS feature is indicated by the Core::X86::Cpuid::FeatureExtIdEcx[IBS].

Instruction execution performance is profiled by tagging one micro-Op associated with an instruction. Instructions that decode to more than one micro-Op return different performance data depending upon which micro-Op associated with the instruction is tagged. These micro-Ops are associated with the RIP of the next instruction.

In this mode, the CPU Profiler uses the IBS HW supported by the AMD x86-based processor to observe the effect of instructions on the processor and on the memory subsystem. In IBS, the hardware events are linked with the instruction that caused them. Also, the hardware events are used by the CPU Profiler to derive various metrics, such as data cache latency.

IBS is supported starting from the AMD processor family 10h.
L3 Cache Performance Monitor Counters (L3PMC)

A Core Complex (CCX) is a group of CPU cores that share L3 cache resources. All the cores in a CCX share a single L3 cache. L3PMCs are available for AMD “Zen”-based processors to monitor the performance of L3 resources. For more information, refer the respective PPR for the processor.

Data Fabric Performance Monitor Counters (DFPMC)

For AMD “Zen”-based processors, DFPMCs are available to monitor the performance of Data Fabric resources. For more information, refer the respective Processor Programming Reference (PPR) for the processor.

12.2.2 Profiling Concepts

Sampling

Sampling profilers work based on the logic that the part of a program that consumes most of the time (or that triggers the most occurrence of the sampling event) have a larger number of samples. This is because they have a higher probability of being executed while samples are being taken by the CPU Profiler.

Sampling Interval

The time between the collection of every two samples is the Sampling Interval. For example, in TBP, if the time interval is 1 millisecond, then roughly 1,000 TBP samples are being collected every second for each processor core.

The purpose of a sampling interval depends on the resource used as the sampling event:

- OS timer — the sampling interval is in milliseconds.
- PMC events — the sampling interval is the number of occurrences of that sampling event.
- IBS — the number of processed instructions after which it will be tagged.

Smaller sampling interval increases the number of samples collected and the data collection overhead. Since, the profile data is collected on the same system in which the workload is running, more frequent sampling increases the intrusiveness of profiling. A very small sampling interval also can cause system instability.

Sampling-point: When a sampling-point occurs upon the expiry of the sampling-interval for a sampling-event, various profile data, such as Instruction Pointer, Process Id, Thread Id, and Call-stack will be collected by the interrupt handler.

Event-Counter Multiplexing

If the number of the monitored PMC events is less than or equal to the number of available performance counters, then each event can be assigned to a counter and monitored 100% of the time. In a single-profile measurement, if the number of monitored events is larger than the number of available counters, the CPU Profiler time-shares the available HW PMC counters. This is called event counter multiplexing. It helps monitor more events and decreases the actual number of samples for each event and thus, reduces the data accuracy. The CPU Profiler auto-scales the sample counts to
compensate for this event counter multiplexing. For example, if an event is monitored 50% of the time, the CPU Profiler scales the number of event samples by factor of 2.

### 12.2.3 Profile Types

The profile types are classified based on the hardware or software sampling events used to collect the profile data.

**Time-Based Profile (TBP)**

In this profile, the profile data is periodically collected based on the specified OS timer interval. It is used to identify the hotspots of the profiled applications.

**Event-Based Profile (EBP)**

In this profile, the CPU Profiler uses the PMCs to monitor the various micro-architectural events supported by the AMD x86-based processor. It helps to identify the CPU and memory related performance issues in the profiled applications. The CPU Profiler provides several predefined EBP profile configurations. To analyze an aspect of the profiled application (or system), a specific set of relevant events are grouped and monitored together. The CPU Profiler provides a list of predefined event configurations, such as Assess Performance and Investigate Branching. You can select any of these predefined configurations to profile and analyze the runtime characteristics of your application. You also can create their custom configurations of events to profile.

In this profile mode, a delay called skid occurs between the time at which the sampling interrupt occurs and the time at which the sampled instruction address is collected. This skid distributes the samples in the neighborhood near the actual instruction that triggered a sampling interrupt. This produces an inaccurate distribution of samples and events are often attributed to the wrong instructions.

**Instruction-Based Sampling (IBS)**

In this profile, the CPU Profiler uses the IBS HW supported by the AMD x86-based processor to observe the effect of instructions on the processor and on the memory subsystem. In IBS, HW events are linked with the instruction that caused them. Also, HW events used by the CPU Profiler to derive various metrics, such as data cache latency.

**Custom Profile**

This profile allows a combination of HW PMC events, OS timer, and IBS sampling events.
### 12.2.4 Predefined Core PMC Events

Some of the Core Performance events of AMD “Zen” processors are listed in the following table:

**Table 46. Predefined Core PMC Events**

<table>
<thead>
<tr>
<th>Event Id, Unit-mask</th>
<th>Event Abbreviation</th>
<th>Name and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x76, 0x00</td>
<td>CYCLES_NOT_IN_HALT</td>
<td>CPU clock cycles not halted The number of CPU cycles when the thread is not in halt state.</td>
</tr>
<tr>
<td>0xC0, 0x00</td>
<td>RETIRED_INST</td>
<td>Retired Instructions The number of instructions retired from execution. This count includes exceptions and interrupts. Each exception or interrupt is counted as one instruction.</td>
</tr>
<tr>
<td>0xC1, 0x00</td>
<td>RETIRED_MICRO_OPS</td>
<td>Retired Macro Operations The number of macro-ops retired. This count includes all processor activity - instructions, exceptions, interrupts, microcode assists, and so on.</td>
</tr>
<tr>
<td>0xC2, 0x00</td>
<td>RETIRED_BR_INST</td>
<td>Retired Branch Instructions The number of branch instructions retired. This includes all types of architectural control flow changes, including exceptions and interrupts.</td>
</tr>
<tr>
<td>0xC3, 0x00</td>
<td>RETIRED_BR_INST_MISP</td>
<td>Retired Branch Instructions Mispredicted The number of retired branch instructions that were mis-predicted. <em>Note: Only EX direct mis-predicts and indirect target mis-predicts are counted.</em></td>
</tr>
<tr>
<td>0x03, 0x08</td>
<td>RETIRED_SSE_AVX_FLOPS</td>
<td>Retired SSE/AVX Flops The number of retired SSE/AVX flops. The number of events logged per cycle can vary from 0 to 64. This is a large increment per cycle event as it can count more than 15 events per cycle. This count both single precision and double precision FP events.</td>
</tr>
</tbody>
</table>
### Table 46. Predefined Core PMC Events

<table>
<thead>
<tr>
<th>Event Id, Unit-mask</th>
<th>Event Abbreviation</th>
<th>Name and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x29, 0x07</td>
<td>L1_DC_ACCESSSES.ALL</td>
<td>All Data cache accesses&lt;br&gt;The number of load and store ops dispatched to LS unit. This counts the dispatch of single op that performs a memory load, dispatch of single op that performs a memory store, dispatch of a single op that performs a load from and store to the same memory address.</td>
</tr>
<tr>
<td>0x60, 0x10</td>
<td>L2_CACHE_ACCESS.FROM_L1_IC_MISS</td>
<td>L2 cache access from L1 IC miss&lt;br&gt;The L2 cache access requests due to L1 instruction cache misses.</td>
</tr>
<tr>
<td>0x60, 0xC8</td>
<td>L2_CACHE_ACCESS.FROM_L1_DC_MISS</td>
<td>L2 cache access from L1 DC miss&lt;br&gt;The L2 cache access requests due to L1 data cache misses. This also counts hardware and software prefetches.</td>
</tr>
<tr>
<td>0x64, 0x01</td>
<td>L2_CACHE_MISS.FROM_L1_IC_MISS</td>
<td>L2 cache miss from L1 IC miss&lt;br&gt;Counts all the Instruction cache fill requests that misses in L2 cache</td>
</tr>
<tr>
<td>0x64, 0x08</td>
<td>L2_CACHE_MISS.FROM_L1_DC_MISS</td>
<td>L2 cache miss from L1 DC miss&lt;br&gt;Counts all the Data cache fill requests that misses in L2 cache</td>
</tr>
<tr>
<td>0x71, 0x1F</td>
<td>L2_HWPF_HIT_IN_L3</td>
<td>L2 Prefetcher Hits in L3&lt;br&gt;Counts all L2 prefetches accepted by the L2 pipeline which miss the L2 cache and hit the L3.</td>
</tr>
<tr>
<td>0x72, 0x1F</td>
<td>L2_HWPF_MISS_IN_L2_L3</td>
<td>L2 Prefetcher Misses in L3&lt;br&gt;Counts all L2 prefetches accepted by the L2 pipeline which miss the L2 and the L3 caches</td>
</tr>
<tr>
<td>0x64, 0x06</td>
<td>L2_CACHE_HIT.FROM_L1_IC_MISS</td>
<td>L2 cache hit from L1 IC miss&lt;br&gt;Counts all the Instruction cache fill requests that hits in L2 cache.</td>
</tr>
<tr>
<td>0x64, 0x70</td>
<td>L2_CACHE_HIT.FROM_L1_DC_MISS</td>
<td>L2 cache hit from L1 DC miss&lt;br&gt;Counts all the Data cache fill requests that hits in L2 cache.</td>
</tr>
<tr>
<td>0x70, 0x1F</td>
<td>L2_HWPF_HIT_IN_L2</td>
<td>L2 cache hit from L2 HW Prefetch&lt;br&gt;Counts all L2 prefetches accepted by L2 pipeline which hit in the L2 cache</td>
</tr>
</tbody>
</table>
### Table 46. Predefined Core PMC Events

<table>
<thead>
<tr>
<th>Event Id, Unit-mask</th>
<th>Event Abbreviation</th>
<th>Name and Description</th>
</tr>
</thead>
</table>
| 0x43, 0x01          | L1_DEMAND_DC_REFILLS.LOCAL_L2 | L1 demand DC fills from L2
The demand Data Cache (DC) fills from local L2 cache to the core. |
| 0x43, 0x02          | L1_DEMAND_DC_REFILLS.LOCAL_CACHE | L1 demand DC fills from local CCX
The demand Data Cache (DC) fills from same the cache of same CCX or cache of different CCX in the same package (node). |
| 0x43, 0x08          | L1_DEMAND_DC_REFILLS.LOCAL_DRAM | L1 demand DC fills from local Memory
The demand Data Cache (DC) fills from DRAM or IO connected in the same package (node). |
| 0x43, 0x10          | L1_DEMAND_DC_REFILLS.REMOTE_CACHE | L1 demand DC fills from remote cache
The demand Data Cache (DC) fills from cache of CCX in the different package (node). |
| 0x43, 0x40          | L1_DEMAND_DC_REFILLS.REMOTE_DRAM | L1 demand DC fills from remote Memory
The demand Data Cache (DC) fills from DRAM or IO connected in the different package (node). |
| 0x43, 0x5B          | L1_DEMAND_DC_REFILLS.ALL | L1 demand DC refills from all data sources.
The demand Data Cache (DC) fills from all the data sources. |
| 0x60, 0xFF          | L2_REQUESTS.ALL | All L2 cache requests. |
| 0x87, 0x01          | STALLED_CYCLES.BACKEND | Instruction pipe stall
The Instruction Cache pipeline was stalled during this cycle due to back-pressure. |
| 0x87, 0x02          | STALLED_CYCLES.FRONTEND | Instruction pipe stall.
The Instruction Cache pipeline was stalled during this cycle due to upstream queues not providing fetch addresses quickly. |
| 0x84, 0x00          | L1_ITLB_MISSES_L2_HITS | L1 TLB miss L2 TLB hit
The instruction fetches that misses in the L1 Instruction Translation Lookaside Buffer (ITLB) but hit in the L2-ITLB. |
| 0x85, 0x07          | L2_ITLB_MISSES | L1 TLB miss L2 TLB miss
The ITLB reloads originating from page table walker. The table walk requests are made for L1-ITLB miss and L2-ITLB misses. |
### Table 46. Predefined Core PMC Events

<table>
<thead>
<tr>
<th>Event Id, Unit-mask</th>
<th>Event Abbreviation</th>
<th>Name and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x45, 0xFF</td>
<td>L1_DTLB_MISSES</td>
<td>L1 DTLB miss</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The L1 Data Translation Lookaside Buffer (DTLB) misses from load store micro-ops. This event counts both L2-DTLB hit and L2-DTLB miss.</td>
</tr>
<tr>
<td>0x45, 0xF0</td>
<td>L2_DTLB_MISSES</td>
<td>L1 DTLB miss</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops.</td>
</tr>
<tr>
<td>0x47, 0x00</td>
<td>MISALIGNED_LOADS</td>
<td>Misaligned Loads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of misaligned loads. On AMD “Zen 3” core processors, this event counts the 64B (cache-line crossing) and 4K (page crossing) misaligned loads.</td>
</tr>
<tr>
<td>0x52, 0x03</td>
<td>INEFFECTIVE_SW_PF</td>
<td>Ineffective Software Prefetches</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of software prefetches that did not fetch data outside of the processor core. This event counts the Software PREFETCH instruction that saw a match on an already-allocated miss request buffer. Also counts the Software PREFETCH instruction that saw a DC hit.</td>
</tr>
<tr>
<td>0x76, 0x00</td>
<td>CYCLES_NOT_IN_HALT</td>
<td>CPU clock cycles not halted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of CPUcycles when the thread is not in halt state.</td>
</tr>
<tr>
<td>0xC0, 0x00</td>
<td>RETIRED_INST</td>
<td>Retired Instructions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of instructions retired from execution. This count includes exceptions and interrupts. Each exception or interrupt is counted as one instruction.</td>
</tr>
<tr>
<td>0xC1, 0x00</td>
<td>RETIRED_MACRO_OPS</td>
<td>Retired Macro Operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of macro-ops retired. This count includes all processor activity - instructions, exceptions, interrupts, microcode assists, and so on.</td>
</tr>
<tr>
<td>0xC2, 0x00</td>
<td>RETIRED_BR_INST</td>
<td>Retired Branch Instructions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of branch instructions retired. This includes all types of architectural control flow changes, including exceptions and interrupts</td>
</tr>
</tbody>
</table>

**AMD EPYC™ 3rd Generation Processors**

<table>
<thead>
<tr>
<th>Event Id, Unit-mask</th>
<th>Event Abbreviation</th>
<th>Name and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x45, 0xFF</td>
<td>L1_DTLB_MISSES</td>
<td>L1 DTLB miss</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The L1 Data Translation Lookaside Buffer (DTLB) misses from load store micro-ops. This event counts both L2-DTLB hit and L2-DTLB miss.</td>
</tr>
<tr>
<td>0x45, 0xF0</td>
<td>L2_DTLB_MISSES</td>
<td>L1 DTLB miss</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops.</td>
</tr>
<tr>
<td>0x47, 0x00</td>
<td>MISALIGNED_LOADS</td>
<td>Misaligned Loads</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of misaligned loads. On AMD “Zen 3” core processors, this event counts the 64B (cache-line crossing) and 4K (page crossing) misaligned loads.</td>
</tr>
<tr>
<td>0x52, 0x03</td>
<td>INEFFECTIVE_SW_PF</td>
<td>Ineffective Software Prefetches</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of software prefetches that did not fetch data outside of the processor core. This event counts the Software PREFETCH instruction that saw a match on an already-allocated miss request buffer. Also counts the Software PREFETCH instruction that saw a DC hit.</td>
</tr>
<tr>
<td>0x76, 0x00</td>
<td>CYCLES_NOT_IN_HALT</td>
<td>CPU clock cycles not halted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of CPUcycles when the thread is not in halt state.</td>
</tr>
<tr>
<td>0xC0, 0x00</td>
<td>RETIRED_INST</td>
<td>Retired Instructions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of instructions retired from execution. This count includes exceptions and interrupts. Each exception or interrupt is counted as one instruction.</td>
</tr>
<tr>
<td>0xC1, 0x00</td>
<td>RETIRED_MACRO_OPS</td>
<td>Retired Macro Operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of macro-ops retired. This count includes all processor activity - instructions, exceptions, interrupts, microcode assists, and so on.</td>
</tr>
<tr>
<td>0xC2, 0x00</td>
<td>RETIRED_BR_INST</td>
<td>Retired Branch Instructions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The number of branch instructions retired. This includes all types of architectural control flow changes, including exceptions and interrupts</td>
</tr>
</tbody>
</table>
### Table 46. Predefined Core PMC Events

<table>
<thead>
<tr>
<th>Event Id, Unit-mask</th>
<th>Event Abbreviation</th>
<th>Name and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xC3, 0x00</td>
<td>RETIRED_BR_INST_MISP</td>
<td>Retired Branch Instructions Mis-predicted The number of retired branch instructions, that were mis-predicted. Note that only EX direct mis-predicts and indirect target mis-predicts are counted.</td>
</tr>
<tr>
<td>0x03, 0x08</td>
<td>RETIRED_SSE_AVX_FLOPS</td>
<td>Retired SSE/AVX Flops The number of retired SSE/AVX flops. The number of events logged per cycle can vary from 0 to 64. This is large increment per cycle event, since it can count more than 15 events per cycle. This count both single precision and double precision FP events.</td>
</tr>
<tr>
<td>0x29, 0x07</td>
<td>L1_DC_ACCESSSES.ALL</td>
<td>All Data cache accesses The number of load and store ops dispatched to LS unit. This counts the dispatch of single op that performs a memory load, dispatch of single op that performs a memory store, and dispatch of a single op that performs a load from and store to the same memory address.</td>
</tr>
<tr>
<td>0x60, 0x10</td>
<td>L2_CACHE_ACCESS.FROM_L1_IC_MISS</td>
<td>L2 cache access from L1 IC miss The L2 cache access requests due to L1 instruction cache misses.</td>
</tr>
<tr>
<td>0x60, 0xE8</td>
<td>L2_CACHE_ACCESS.FROM_L1_DC_MISS</td>
<td>L2 cache access from L1 DC miss The L2 cache access requests due to L1 data cache misses. This also counts hardware and software prefetches.</td>
</tr>
<tr>
<td>0x64, 0x01</td>
<td>L2_CACHE_MISS.FROM_L1_IC_MISS</td>
<td>L2 cache miss from L1 IC miss Counts all the Instruction cache fill requests that misses in L2 cache.</td>
</tr>
<tr>
<td>0x64, 0x08</td>
<td>L2_CACHE_MISS.FROM_L1_DC_MISS</td>
<td>L2 cache miss from L1 DC miss Counts all the Data cache fill requests that misses in L2 cache.</td>
</tr>
<tr>
<td>0x71, 0xFF</td>
<td>L2_HWPF_HIT_IN_L3</td>
<td>L2 Prefetcher Hits in L3 Counts all L2 prefetches accepted by the L2 pipeline which miss the L2 cache and hit the L3.</td>
</tr>
<tr>
<td>0x72, 0xFF</td>
<td>L2_HWPF_MISS_IN_L2_L3</td>
<td>L2 Prefetcher Misses in L3 Counts all L2 prefetches accepted by the L2 pipeline which miss the L2 and the L3 caches.</td>
</tr>
</tbody>
</table>
### Table 46.  Predefined Core PMC Events

<table>
<thead>
<tr>
<th>Event Id, Unit-mask</th>
<th>Event Abbreviation</th>
<th>Name and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x64, 0x06</td>
<td>L2_CACHE_HIT.FROM_L1_IC_MISS</td>
<td>L2 cache hit from L1 IC miss Counts all the Instruction cache fill requests that hits in L2 cache.</td>
</tr>
<tr>
<td>0x64, 0xF0</td>
<td>L2_CACHE_HIT.FROM_L1_DC_MISS</td>
<td>L2 cache hit from L1 DC miss Counts all the Data cache fill requests that hits in L2 cache.</td>
</tr>
<tr>
<td>0x70, 0xFF</td>
<td>L2_HWPF_HIT_IN_L2</td>
<td>L2 cache hit from L2 HW Prefetch Counts all L2 prefetches accepted by L2 pipeline which hit in the L2 cache.</td>
</tr>
<tr>
<td>0x43, 0x01</td>
<td>L1_DEMAND_DC_REFILLS.LOCAL_L2</td>
<td>L1 demand DC fills from L2 The demand Data Cache (DC) fills from local L2 cache to the core.</td>
</tr>
<tr>
<td>0x43, 0x02</td>
<td>L1_DEMAND_DC_REFILLS.LOCAL_CACHE</td>
<td>L1 demand DC fills from local CCX The demand Data Cache (DC) fills from the L3 cache or L2 in the same CCX.</td>
</tr>
<tr>
<td>0x43, 0x04</td>
<td>L1_DC_REFILLS.EXTERNAL_CACHE_LOCAL</td>
<td>L1 DC fills from local external CCX caches The Data Cache (DC) fills from cache of different CCX in the same package (node).</td>
</tr>
<tr>
<td>0x43, 0x08</td>
<td>L1_DEMAND_DC_REFILLS.LOCAL_DRAM</td>
<td>L1 demand DC fills from local Memory The demand Data Cache (DC) fills from DRAM or IO connected in the same package (node).</td>
</tr>
<tr>
<td>0x43, 0x10</td>
<td>L1_DEMAND_DC_REFILLS.EXTERNAL_CACHE_REMOTE</td>
<td>L1 demand DC fills from remote external cache The demand Data Cache (DC) fills from cache of CCX in the different package (node).</td>
</tr>
<tr>
<td>0x43, 0x40</td>
<td>L1_DEMAND_DC_REFILLS.REMOTE_DRAM</td>
<td>L1 demand DC fills from remote Memory The demand Data Cache (DC) fills from DRAM or IO connected in the different package (node).</td>
</tr>
<tr>
<td>0x43, 0x14</td>
<td>L1_DEMAND_DC_REFILLS.EXTERNAL_CACHE</td>
<td>L1 demand DC fills from external caches The demand Data Cache (DC) fills from cache of different CCX in the same or different package (node).</td>
</tr>
<tr>
<td>0x43, 0x5F</td>
<td>L1_DEMAND_DC_REFILLS.ALL</td>
<td>L1 demand DC refills from all data sources. The demand Data Cache (DC) fills from all the data sources.</td>
</tr>
</tbody>
</table>
### Table 46. Predefined Core PMC Events

<table>
<thead>
<tr>
<th>Event Id, Unit-mask</th>
<th>Event Abbreviation</th>
<th>Name and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x44, 0x01</td>
<td>L1_DC_REFILLS.LOCAL_L2</td>
<td>L1 DC fills from local L2 cache to the core.</td>
</tr>
<tr>
<td>0x44, 0x02</td>
<td>L1_DC_REFILLS.LOCAL_CACHE</td>
<td>L1 DC fills from local CCX cache in the same CCX or L3 cache that belongs to the same CCX.</td>
</tr>
<tr>
<td>0x44, 0x08</td>
<td>L1_DC_REFILLS.LOCAL_DRAM</td>
<td>L1 DC fills from local Memory in the same package (node).</td>
</tr>
<tr>
<td>0x44, 0x04</td>
<td>L1_DC_REFILLS.EXTERNAL_CACHE_LOCAL</td>
<td>L1 DC fills from local external CCX caches in the same package (node).</td>
</tr>
<tr>
<td>0x44, 0x10</td>
<td>L1_DC_REFILLS.EXTERNAL_CACHE_REMOTE</td>
<td>L1 DC fills from remote external CCX caches in the different package (node).</td>
</tr>
<tr>
<td>0x44, 0x40</td>
<td>L1_DC_REFILLS.REMOTE_DRAM</td>
<td>L1 DC fills from remote Memory in the different package (node).</td>
</tr>
<tr>
<td>0x44, 0x14</td>
<td>L1_DC_REFILLS.EXTERNAL_CACHE</td>
<td>L1 DC fills from local external CCX caches in the same or different package (node).</td>
</tr>
<tr>
<td>0x44, 0x48</td>
<td>L1_DC_REFILLS.DRAM</td>
<td>L1 DC fills from local Memory in the same or different package (node).</td>
</tr>
<tr>
<td>0x44, 0x50</td>
<td>L1_DC_REFILLS.REMOTE_NODE</td>
<td>L1 DC fills from remote node in the different package (node) or the DRAM / IO connected in the different package (node).</td>
</tr>
<tr>
<td>0x44, 0x03</td>
<td>L1_DC_REFILLS.LOCAL_CACHE_L2_L3</td>
<td>L1 DC fills from same CCX in the different package (node) or the DRAM / IO connected in the different package (node).</td>
</tr>
<tr>
<td>0x44, 0x5F</td>
<td>L1_DC_REFILLS.ALL</td>
<td>L1 DC fills from all the data sources in the different package (node) or the DRAM / IO connected in the different package (node).</td>
</tr>
<tr>
<td>Event Id, Unit-mask</td>
<td>Event Abbreviation</td>
<td>Name and Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>0x60, 0xFF</td>
<td>L2_REQUESTS.ALL</td>
<td>All L2 cache requests.</td>
</tr>
</tbody>
</table>
| 0x87, 0x01          | STALLED_CYCLES.BACKEND | Instruction pipe stall  
The Instruction Cache pipeline was stalled during this cycle due to back-pressure. |
| 0x87, 0x02          | STALLED_CYCLES.FRONTEND | Instruction pipe stall  
The Instruction Cache pipeline was stalled during this cycle due to upstream queues not providing fetch addresses quickly. |
| 0x84, 0x00          | L1_ITLB_MISS_L2_HITS | L1 TLB miss L2 TLB hit  
The instruction fetches that misses in the L1 Instruction Translation Lookaside Buffer (ITLB) but hit in the L2-ITLB. |
| 0x85, 0x07          | L2_ITLB_MISS        | L1 TLB miss L2 TLB miss  
The ITLB reloads originating from page table walker. The table walk requests are made for L1-ITLB miss and L2-ITLB misses. |
| 0x45, 0xFF          | L1_DTLB_MISS        | L1 DTLB miss  
The L1 Data Translation Lookaside Buffer (DTLB) misses from load store micro-ops. This event counts both L2-DTLB hit and L2-DTLB miss. |
| 0x45, 0xF0          | L2_DTLB_MISS        | L1 DTLB miss  
The L2 Data Translation Lookaside Buffer (DTLB) missed from load store micro-ops. |
| 0x78, 0xFF          | ALL_TLB_FLUSH      | All TLB flushes |
| 0x47, 0x03          | MISALIGNED_LOADS   | Misaligned Loads  
The number of misaligned loads. On AMD “Zen 3” core processors, this event counts the 64B (cache-line crossing) and 4K (page crossing) misaligned loads. |
| 0x52, 0x03          | INEFFECTIVE_SW_PF  | Ineffective Software Prefetches  
The number of software prefetches that did not fetch data outside of the processor core. This event counts the Software PREFETCH instruction that saw a match on an already allocated miss request buffer. Also counts the Software PREFETCH instruction that saw a DC hit. |
The following table shows the CPU performance metrics:

**Table 47. Core CPU Metrics**

<table>
<thead>
<tr>
<th>CPU Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Effective Frequency</td>
<td>Core Effective Frequency (without halted cycles) over the sampling period, reported in GHz. The metric is based on APERF and MPERF MSRs. MPERF is incremented by the core at the P0 state frequency while the core is in C0 state. APERF is incremented in proportion to the actual number of core cycles while the core is in C0 state.</td>
</tr>
<tr>
<td>IPC</td>
<td>Instruction Retired Per Cycle (IPC) is the average number of instructions retired per cycle. This is measured using Core PMC events PMCx0C0 [Retired Instructions] and PMCx076 [CPU Clocks not Halted]. These PMC events are counted in both OS and User mode.</td>
</tr>
<tr>
<td>CPI</td>
<td>Cycles Per Instruction Retired (CPI) is the multiplicative inverse of IPC metric. This is one of the basic performance metrics indicating how cache misses, branch mis-predictions, memory latencies, and other bottlenecks are affecting the execution of an application. Lower CPI value is better.</td>
</tr>
<tr>
<td>L1_DC_REFILLS.ALL (PTI)</td>
<td>The number of demand data cache (DC) fills per thousand retired instructions. These demand DC fills are from all the data sources like Local L2/L3 cache, remote caches, local memory, and remote memory.</td>
</tr>
<tr>
<td>L1_DC_MISSES (PTI)</td>
<td>The number of L2 cache access requests due to L1 data cache misses, per thousand retired instructions. This L2 cache access requests also includes the hardware and software prefetches.</td>
</tr>
<tr>
<td>L1_DC_ACCESS_RATE</td>
<td>The DC access rate is the number of DC accesses divided by the total number of retired instructions</td>
</tr>
<tr>
<td>L1_DC_MISS_RATE</td>
<td>The DC miss rate is the number of DC misses divided by the total number of retired instructions.</td>
</tr>
<tr>
<td>L1_DC_MISS_RATIO</td>
<td>The DC miss ratio is the number of DC misses divided by the total number of DC accesses.</td>
</tr>
</tbody>
</table>

### 12.2.5 IBS Derived Events

AMD uProf translates the IBS information produced by the hardware into derived event sample counts that resemble EBP sample counts. All the IBS-derived events contain IBS in the event name and abbreviation. Although IBS-derived events and sample counts look similar to the EBP events and sample counts, the source and sampling basis for the IBS event information are different.

Arithmetic should never be performed between IBS derived event sample counts and EBP event sample counts. It is not meaningful to directly compare the number of samples taken for events that represent the same hardware condition. For example, fewer IBS DC miss samples is not necessarily better than a larger quantity of EBP DC miss samples.
The following table shows the IBS fetch events:

**Table 48. IBS Fetch Events**

<table>
<thead>
<tr>
<th>IBS Fetch Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All IBS Fetch Samples</td>
<td>The number of all the IBS fetch samples. This derived event counts the number of all the IBS fetch samples that were collected including IBS-killed fetch samples.</td>
</tr>
<tr>
<td>IBS Fetch Killed</td>
<td>The number of IBS sampled fetches that were killed fetches. A fetch operation is killed if the fetch did not reach ITLB or IC access. The number of killed fetch samples is not generally useful for analysis and are filtered out in other derived IBS fetch events (except Event Select 0xF000 which counts all IBS fetch samples including IBS killed fetch samples).</td>
</tr>
<tr>
<td>IBS Fetch Attempted</td>
<td>The number of IBS sampled fetches that were not killed fetch attempts. This derived event measures the number of useful fetch attempts and does not include the number of IBS killed fetch samples. This event should be used to compute ratios such as the ratio of IBS fetch IC misses to attempted fetches. The number of attempted fetches should equal the sum of the number of completed fetches and the number of aborted fetches.</td>
</tr>
<tr>
<td>IBS Fetch Completed</td>
<td>The number of completed IBS sampled fetches. A fetch is completed if the attempted fetch delivers instruction data to the instruction decoder. Although the instruction data was delivered, it may still not be used. For example, the instruction data may have been on the “wrong path” of an incorrectly predicted branch.</td>
</tr>
<tr>
<td>IBS Fetch Aborted</td>
<td>The number of IBS sampled fetches that aborted. An attempted fetch is aborted if it did not complete and deliver instruction data to the decoder. An attempted fetch may abort at any point in the process of fetching instruction data. An abort may be due to a branch redirection as the result of a mispredicted branch. The number of IBS aborted fetch samples is a lower bound on the number of unsuccessful, speculative fetch activity. It is a lower bound as the instruction data delivered by completed fetches may not be used.</td>
</tr>
<tr>
<td>IBS ITLB hit</td>
<td>The number of IBS attempted fetch samples where the fetch operation initially hit in the L1 ITLB (Instruction Translation Lookaside Buffer).</td>
</tr>
<tr>
<td>IBS L1 ITLB misses (and L2 ITLB hits)</td>
<td>The number of IBS attempted fetch samples where the fetch operation initially missed in the L1 ITLB and hit in the L2 ITLB.</td>
</tr>
<tr>
<td>IBS L1 L2 ITLB miss</td>
<td>The number of IBS attempted fetch samples where the fetch operation initially missed in both the L1 ITLB and the L2 ITLB.</td>
</tr>
<tr>
<td>IBS instruction cache misses</td>
<td>The number of IBS attempted fetch samples where the fetch operation initially missed in the IC (instruction cache).</td>
</tr>
<tr>
<td>IBS instruction cache hit</td>
<td>The number of IBS attempted fetch samples where the fetch operation initially hit in the IC.</td>
</tr>
<tr>
<td>IBS 4K page translation</td>
<td>The number of IBS attempted fetch samples where the fetch operation produced a valid physical address (that is, address translation completed successfully) and used a 4-KByte page entry in the L1 ITLB.</td>
</tr>
</tbody>
</table>
Table 48. IBS Fetch Events

<table>
<thead>
<tr>
<th>IBS Fetch Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS 2M page translation</td>
<td>The number of IBS attempted fetch samples where the fetch operation produced a valid physical address (that is, address translation completed successfully) and used a 2 MB page entry in the L1 ITLB.</td>
</tr>
<tr>
<td>IBS fetch latency</td>
<td>The total latency of all IBS attempted fetch samples. Divide the total IBS fetch latency by the number of IBS attempted fetch samples to obtain the average latency of the attempted fetches that were sampled.</td>
</tr>
<tr>
<td>IBS fetch L2 cache miss</td>
<td>The instruction fetch missed in the L2 Cache.</td>
</tr>
<tr>
<td>IBS ITLB refill latency</td>
<td>The number of cycles when the fetch engine is stalled for an ITLB reload for the sampled fetch. If there is no reload, the latency will be 0.</td>
</tr>
</tbody>
</table>

The following table lists the IBS op events:

Table 49. IBS Op Events

<table>
<thead>
<tr>
<th>IBS Op Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All IBS op samples</td>
<td>The number of all IBS op samples that were collected. These op samples may be branch ops, resync ops, ops that perform load/store operations, or undifferentiated ops (for example, the ops that perform arithmetic and logical operations). IBS collects data for the retired ops. No data is collected for ops that are aborted due to pipeline flushes. Thus, all sampled ops are architecturally significant and contribute to the successful forward progress of executing programs.</td>
</tr>
<tr>
<td>IBS tag-to-retire cycles</td>
<td>The total number of tag-to-retire cycles across all IBS op samples. The tag-to-retire time of an op is the number of cycles from when the op was tagged (selected for sampling) to when the op retired.</td>
</tr>
<tr>
<td>IBS completion-to-retire cycles</td>
<td>The total number of completion-to-retire cycles across all the IBS op samples. The completion-to-retire time of an op is the number of cycles from when the op completed to when the op retired.</td>
</tr>
<tr>
<td>IBS branch op</td>
<td>The number of IBS retired branch op samples. A branch operation is a change in program control flow and includes unconditional and conditional branches, subroutine calls, and subroutine returns. Branch ops are used to implement AMD64 branch semantics.</td>
</tr>
<tr>
<td>IBS mispredicted branch op</td>
<td>The number of IBS samples for retired branch operations that were mispredicted. This event should be used to compute the ratio of mispredicted branch operations to all branch operations.</td>
</tr>
<tr>
<td>IBS taken branch op</td>
<td>The number of IBS samples for retired branch operations that were taken branches.</td>
</tr>
<tr>
<td>IBS mispredicted taken branch op</td>
<td>The number of IBS samples for retired branch operations that were mispredicted taken branches.</td>
</tr>
</tbody>
</table>
### Table 49: IBS Op Events

<table>
<thead>
<tr>
<th>IBS Op Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS return op</td>
<td>The number of IBS retired branch op samples where the operation was a subroutine return. These samples are a subset of all IBS retired branch op samples.</td>
</tr>
<tr>
<td>IBS mispredicted return op</td>
<td>The number of IBS retired branch op samples where the operation was a mispredicted subroutine return. This event should be used to compute the ratio of mispredicted returns to all subroutine returns.</td>
</tr>
<tr>
<td>IBS resync op</td>
<td>The number of IBS resync op samples. A resync op is only found in certain micro-coded AMD64 instructions and causes a complete pipeline flush.</td>
</tr>
<tr>
<td>IBS all load store ops</td>
<td>The number of IBS op samples for ops that perform either a load and/or store operation. An AMD64 instruction may be translated into one (single fast path), two (double fast path), or several (vector path) ops. Each op may perform a load operation, a store operation, or both (each to the same address). Some op samples attributed to an AMD64 instruction may perform a load/store operation while other op samples attributed to the same instruction may not. Further, some branch instructions perform load/store operations. Thus, a mix of op sample types may be attributed to a single AMD64 instruction depending upon the ops that are issued from the AMD64 instruction and the op types.</td>
</tr>
<tr>
<td>IBS load ops</td>
<td>The number of IBS op samples for ops that perform a load operation.</td>
</tr>
<tr>
<td>IBS store ops</td>
<td>The number of IBS op samples for ops that perform a store operation.</td>
</tr>
<tr>
<td>IBS L1 DTLB hit</td>
<td>The number of IBS op samples where either a load or store operation initially hit in the L1 DTLB (data translation lookaside buffer).</td>
</tr>
<tr>
<td>IBS L1 DTLB misses L2 hits</td>
<td>The number of IBS op samples where either a load or store operation initially missed in the L1 DTLB and hit in the L2 DTLB.</td>
</tr>
<tr>
<td>IBS L1 and L2 DTLB misses</td>
<td>The number of IBS op samples where either a load or store operation initially missed in both the L1 DTLB and the L2 DTLB.</td>
</tr>
<tr>
<td>IBS data cache misses</td>
<td>The number of IBS op samples where either a load or store operation initially missed in the data cache (DC).</td>
</tr>
<tr>
<td>IBS data cache hits</td>
<td>The number of IBS op samples where either a load or store operation initially hit in the data cache (DC).</td>
</tr>
<tr>
<td>IBS misaligned data access</td>
<td>The number of IBS op samples where either a load or store operation caused a misaligned access (that is, the load or store operation crossed a 128-bit boundary).</td>
</tr>
<tr>
<td>IBS bank conflict on load op</td>
<td>The number of IBS op samples where either a load or store operation caused a bank conflict with a load operation.</td>
</tr>
<tr>
<td>IBS bank conflict on store op</td>
<td>The number of IBS op samples where either a load or store operation caused a bank conflict with a store operation.</td>
</tr>
<tr>
<td>IBS store-to-load forwarded</td>
<td>The number of IBS op samples where data for a load operation was forwarded from a store operation.</td>
</tr>
</tbody>
</table>
## Table 49. IBS Op Events

<table>
<thead>
<tr>
<th>IBS Op Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS store-to-load cancelled</td>
<td>The number of IBS op samples where data forwarding to a load operation from a store was canceled.</td>
</tr>
<tr>
<td>IBS UC memory access</td>
<td>The number of IBS op samples where a load or store operation accessed uncacheable (UC) memory.</td>
</tr>
<tr>
<td>IBS WC memory access</td>
<td>The number of IBS op samples where a load or store operation accessed write combining (WC) memory.</td>
</tr>
<tr>
<td>IBS locked operation</td>
<td>The number of IBS op samples where a load or store operation was a locked operation.</td>
</tr>
<tr>
<td>IBS MAB hit</td>
<td>The number of IBS op samples where a load or store operation hit an already allocated entry in the Miss Address Buffer (MAB).</td>
</tr>
<tr>
<td>IBS L1 DTLB 4K page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address and a 4 KB page entry in the L1 DTLB was used for address translation.</td>
</tr>
<tr>
<td>IBS L1 DTLB 2M page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address and a 2 MB page entry in the L1 DTLB was used for address translation.</td>
</tr>
<tr>
<td>IBS L1 DTLB 1G page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address and a 1 GB page entry in the L1 DTLB was used for address translation.</td>
</tr>
<tr>
<td>IBS L2 DTLB 4K page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit the L2 DTLB, and used a 4 KB page entry for address translation.</td>
</tr>
<tr>
<td>IBS L2 DTLB 2M page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit the L2 DTLB, and used a 2 MB page entry for address translation.</td>
</tr>
<tr>
<td>IBS L2 DTLB 1G page</td>
<td>The number of IBS op samples where a load or store operation produced a valid linear (virtual) address, hit the L2 DTLB, and used a 1 GB page entry for address translation.</td>
</tr>
<tr>
<td>IBS data cache miss load latency</td>
<td>The total DC miss load latency (in processor cycles) across all the IBS op samples that performed a load operation and missed in the data cache. The miss latency is the number of clock cycles from when the data cache miss was detected to when data was delivered to the core. Divide the total DC miss load latency by the number of data cache misses to obtain the average DC miss load latency.</td>
</tr>
<tr>
<td>IBS load resync</td>
<td>The Load Resync.</td>
</tr>
</tbody>
</table>
### Table 49. IBS Op Events

<table>
<thead>
<tr>
<th>IBS Op Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBS Northbridge local</td>
<td>The number of IBS op samples where a load operation was serviced from the local processor. Northbridge IBS data is only valid for load operations that miss in both the L1 data cache and the L2 data cache. If a load operation crosses a cache line boundary, then the IBS data reflects the access to the lower cache line.</td>
</tr>
<tr>
<td>IBS Northbridge remote</td>
<td>The number of IBS op samples where a load operation was serviced from a remote processor.</td>
</tr>
<tr>
<td>IBS Northbridge local L3</td>
<td>The number of IBS op samples where a load operation was serviced by the local L3 cache.</td>
</tr>
<tr>
<td>IBS Northbridge local core L1 or L2 cache</td>
<td>The number of IBS op samples where a load operation was serviced by a cache (L1 data cache or L2 cache) belonging to a local core which is a sibling of the core making the memory request.</td>
</tr>
<tr>
<td>IBS Northbridge local core L1, L2, L3 cache</td>
<td>The number of IBS op samples where a load operation was serviced by a remote L1 data cache, L2 cache, or L3 cache after traversing one or more coherent HyperTransport links.</td>
</tr>
<tr>
<td>IBS Northbridge local DRAM</td>
<td>The number of IBS op samples where a load operation was serviced by local system memory (local DRAM through the memory controller).</td>
</tr>
<tr>
<td>IBS Northbridge remote DRAM</td>
<td>The number of IBS op samples where a load operation was serviced by remote system memory (after traversing one or more coherent HyperTransport links and through a remote memory controller).</td>
</tr>
<tr>
<td>IBS Northbridge local APIC MMIO Config PCI</td>
<td>The number of IBS op samples where a load operation was serviced from local MMIO, configuration or PCI space, or from the local APIC.</td>
</tr>
<tr>
<td>IBS Northbridge remote APIC MMIO Config PCI</td>
<td>The number of IBS op samples where a load operation was serviced from remote MMIO, configuration, or PCI space.</td>
</tr>
<tr>
<td>IBS Northbridge cache modified state</td>
<td>The number of IBS op samples where a load operation was serviced from local or remote cache and the cache hit state was the Modified (M) state.</td>
</tr>
<tr>
<td>IBS Northbridge cache owned state</td>
<td>The number of IBS op samples where a load operation was serviced from local or remote cache and the cache hit state was the Owned (O) state.</td>
</tr>
<tr>
<td>IBS Northbridge local cache latency</td>
<td>The total data cache miss latency (in processor cycles) for load operations that were serviced by the local processor.</td>
</tr>
<tr>
<td>IBS Northbridge remote cache latency</td>
<td>The total data cache miss latency (in processor cycles) for load operations that were serviced by a remote processor.</td>
</tr>
</tbody>
</table>
12.3 Useful URLs

For the processor specific PMC events and their descriptions, refer the following AMD developer documents:

- Processor Programming Reference (PPR) for AMD Processors (https://developer.amd.com/resources/developer-guides-manuals/)
- Software Optimization Guide for AMD Family 17h Processors (https://developer.amd.com/wordpress/media/2013/12/55723_3_00.ZIP)